Università di Pisa

Progetto dell'Applicazione di Monitoraggio della Rete Fognaria Urbana

Lavoro svolto all'interno del progetto

Interreg/ADAPT

A cura dei professori
Marco Cococcioni, Francesco Marcelloni e Pietro Ducange

Indice

1 Introduzione **3**

2 Descrizione dell'applicazione **3**

2.1 Front-end (interfaccia grafica) 3

2.2 Database NOSQL (MongoDB) 4

2.3 Back-end 4

2.4 Maggiori dettagli sull'applicazione nel suo insieme 4

2.5 Ringraziamenti 5

3 Progettazione **6**

3.1 Requisiti Funzionali 6

3.2 Requisiti Non-Funzionali 8

3.3 Attori del sistema 8

3.4 Casi d'uso 9

3.5 Analisi delle classi 10

3.6 Modellazione Dati 11

3.6.1 *userData* collection 11

3.6.2 *drainageData* collection 13

4 Simulatore di campioni per le giunzioni **17**

4.1 Il Simulatore 17

5 Implementazione del database MongoDB **20**

5.1 Descrizione delle classi 20

5.1.1 Visualizzazione grafica 20

5.1.2 Rappresentazione degli oggetti 21

5.1.3 Interfaccia verso server MySQL 22

5.1.4 Interfaccia verso server MongoDB 23

5.1.5 Configurazione 23

5.2 Pipeline di aggregazione 23

5.2.1 Media, minimo e massimo in un dato intervallo temporale . . 23

5.2.2 Media, minimo e massimo di un dato mese dell'anno 24

5.2.3 Media, minimo e massimo di un dato anno 25
5.3 Indici MongoDB 27
5.3.1 Indice sui campi idLuogo, dataRiferimento 27
5.3.2 Indice sui campi idLuogo, timestampAggiornamento 29
5.3.3 Indice sul campo citta 30
5.4 Replicazione del database 32
5.5 Sharded cluster 33

6 Applicazione lato Front-End **34**
6.1 Login 34
6.2 News 36
6.3 Elenco Giunzioni 37
6.4 Visualizzazione dati giunzione 37
6.4.1 Dati recenti 38
6.4.2 Dati storici giornalieri 39
6.4.3 Dati storici mensili 40
6.4.4 Dati storici annuali 41
6.5 Pluviometri 42
6.6 Mappe e dati statistici 43
6.7 Area Privata 43

7 Progettazione del Backend **47**
7.1 Caratteristiche dei dati di previsione della pioggia fornite da LAMMA 47
7.1.1 Descrizioni delle nuove previsioni, ogni 12 ore (2020-) 47
7.1.2 Tipologia di dati forniti da LAMMA: forecasts e hindcasts . . 48
7.1.3 Previsioni anni passati (2015-2019) 48

8 Conclusioni e Sviluppi Futuri **50**

1 Introduzione

Il progetto ADAPT si propone di effettuare il monitoraggio del sistema fognario unitario in ambito urbano per le città di Pisa e di Marina di Cecina.

Gli stakeholder del progetto sono:

- l'azienda Acque spa di Pisa
- l'azienda ASA di Livorno

Nell'ambito di tale progetto si è deciso di progettare e realizzare il prototipo di una applicazione in grado di monitorare i livelli dei pozzetti della rete fognaria (in linguaggio modellistico vengono chiamate *junctions*).

2 Descrizione dell'applicazione

L'applicazione che dovrà essere progettata e prototipata è formata da:

- un front-end (interfaccia grafica)
- un database NOSQL scalabile basato su documenti (MongoDB)
- un back-end

2.1 Front-end (interfaccia grafica)

Scopo dell'interfaccia grafica di front-end è quello di supportare i tecnici dei due soggetti stakeholder del progetto ADAPT (l'azienda Acque spa e l'azienda ASA spa) nel monitoraggio e nell'analisi del sistema di drenaggio delle acque bianche e delle acque nere della città di Pisa e Marina di Cecina, al fine di rilevare possibili situazioni di pericolo di allagamento in specifiche zone delle due città.

Si è scelto di implementare queste funzioni come una applicazione stand-alone programmata in Java.

2.2 Database NOSQL (MongoDB)

Scopo del database è quello di storicizzare le altezze di pioggia misurate dai dati meteo resi disponibili dal Servizio Idrologico Regionale (SIR) e dalle stazione meteo installate dai Gestori nelle due città, le misure di portata dei sensori installati nelle sezioni di chiusura idraulica dei bacini analizzati e i livelli idrici dei sensori installati nelle giunzioni più critiche e/o significative.

Si è scelto di utilizzare il database MongoDB, in quanto *document-oriented* e *column-oriented*.

2.3 Back-end

Scopo del back-end è quello di elaborare le previsioni di pioggia fornite da LAMMA ed effettuare una simulazione (mediante software SWMM5) del deflusso dell'acqua piovana all'interno della rete di raccolta di Pisa e di Marina di Cecina.

Si è scelto di implementare questa parte del software in Matlab.

2.4 Maggiori dettagli sull'applicazione nel suo insieme

L'applicazione ha lo scopo di supportare l'utente incaricato del monitoraggio e dell'analisi del sistema di drenaggio delle acque bianche e delle acque nere della città selezionata, al fine di rilevare possibili situazioni di allerta.

Ci sono due componenti principali che devono essere geolocalizzati e monitorati:

- Pluviometri
- Giunzioni

Per ogni componente, l'applicazione deve mostrare tutte le relative informazioni come posizione, seriale del sensore e data di installazione.
Inoltre, per ogni giunzione deve mostrare:

- I dati misurati dal sensore fino alla fine del giorno precedente

- I dati simulati fino alla fine del giorno corrente
- I dati simulati relativi alle previsioni meteo delle successive 48 ore

Per quanto riguarda i pluviometri, verranno mostrati solo i dati misurati dai sensori fino al giorno precedente.

I dati simulati saranno ottenuti dal simulatore SWMM5 (Storm Water Management Model) che prenderà come input le previsioni del livello di pioggia fornite dal consorzio LAMMA e utilizzerà il modello di drenaggio della rete fognaria.

2.5 Ringraziamenti

La progettazione della presente applicazione è stata realizzata con il prezioso supporto degli studenti Sara Lotano, Lorenzo Simi, Filippo Storniolo e Matteo Suffredini.

3 Progettazione

3.1 Requisiti Funzionali

- L'applicazione deve associare ad ogni giunzione **dati simulati** e, se presenti, **dati misurati**.

- L'applicazione deve mostrare, mediante un grafico, il livello dell'acqua misurato dai sensori. Il grafico sarà composto da due diversi intervalli temporali: *passato* e *futuro*.
  L'intervallo temporale *passato* conterrà:

  - per i **Pluviometri**, i dati misurati dai sensori
  - per le **Giunzioni**, i dati misurati, se presenti, oppure quelli simulati

  L'intervallo temporale *futuro*, sarà presente solamente nel grafico delle giunzioni e conterrà i dati simulati per le prossime 48 ore.

- Per tutti i componenti, l'applicazione deve mostrare i dati storici considerando un intervallo temporale specificato dall'utente.

- L'applicazione deve evidenziare la mancanta trasmissione dei campioni da parte di un sensore.

- L'applicazione deve permettere un confronto analitico tra i dati misurati e quelli simulati.

- L'applicazione deve mostrare dati statistici quali media, minimo e massimo per ogni componente.

- L'applicazione deve prevedere quattro stati di allerta: rosso, arancione, giallo e verde, le cui soglie devono essere configurabili. Si precisa che allerta verde vuol dire *nessuna allerta* (ovvero siamo nella fascia di normale funzionamento). Pertanto il livello di allerta rosso è quello di allerta massimo.

- L'applicazione deve mostrare una vista sintetica dello stato di allerta per ogni giunzione, basandosi sui dati previsti per le prossime 48 ore.

- L'applicazione deve prevedere il ruolo di **Amministratore**, il quale deve essere in grado di:
  - importare file aggiornati relativi al modello matematico, a seguito di modifiche e/o tarature
  - configurare le soglie per i livelli di allerta
  - modificare e configurare i profili personali
  - inserire nuovi utenti

3.2 Requisiti Non-Funzionali

- **Scalabilità**: il sistema deve essere in grado di gestire un'elevata mole di dati.
- **Tolleranza ai guasti**: il sistema deve gestire i guasti con l'uso di repliche distribuite su un cluster di server.
- **Consistenza**: la politica adottata è quella di Eventual Consistency tra l'applicazione ed il cluster.
- **Sicurezza**: il sistema deve memorizzare le password applicandovi un algoritmo di crittografia.

3.3 Attori del sistema

I principali attori del sistema sono:

- **Utente**: può visualizzare tutti i dati.
- **Amministratore**: può modificare i campi associati ad una giunzione o ad un pluviometro, importare nuovi dati, modificare i propri dati personali e inserire un nuovo utente.

3.4 Casi d'uso

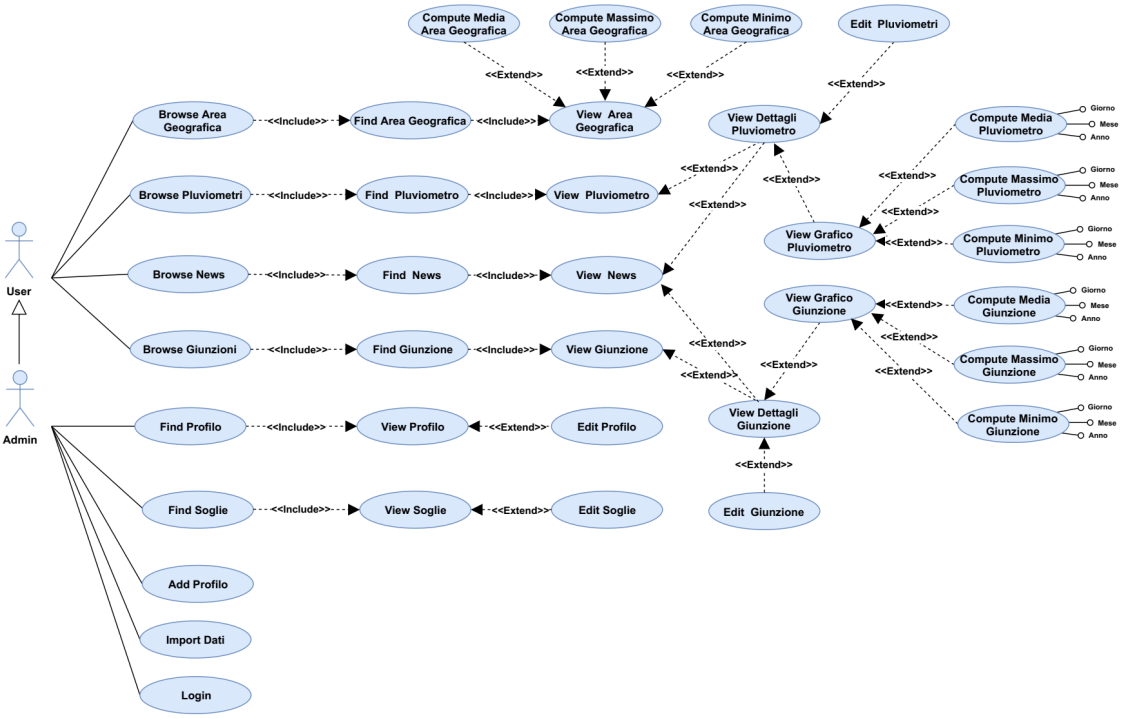


Figura 1: Casi d'uso

3.5 Analisi delle classi

Le principali strutture dati identificate nel sistema sono le seguenti:

- **Campione**, rappresenta la serie di campioni giornalieri relativi ad un sensore.
- **SensoreGiunzione**, rappresenta le informazioni relative ad una giunzione.
- **SensorePluviometro**, rappresenta le informazioni relative ad un pluviometro.
- **Admin**, rappresenta le informazioni relative all'amministratore.

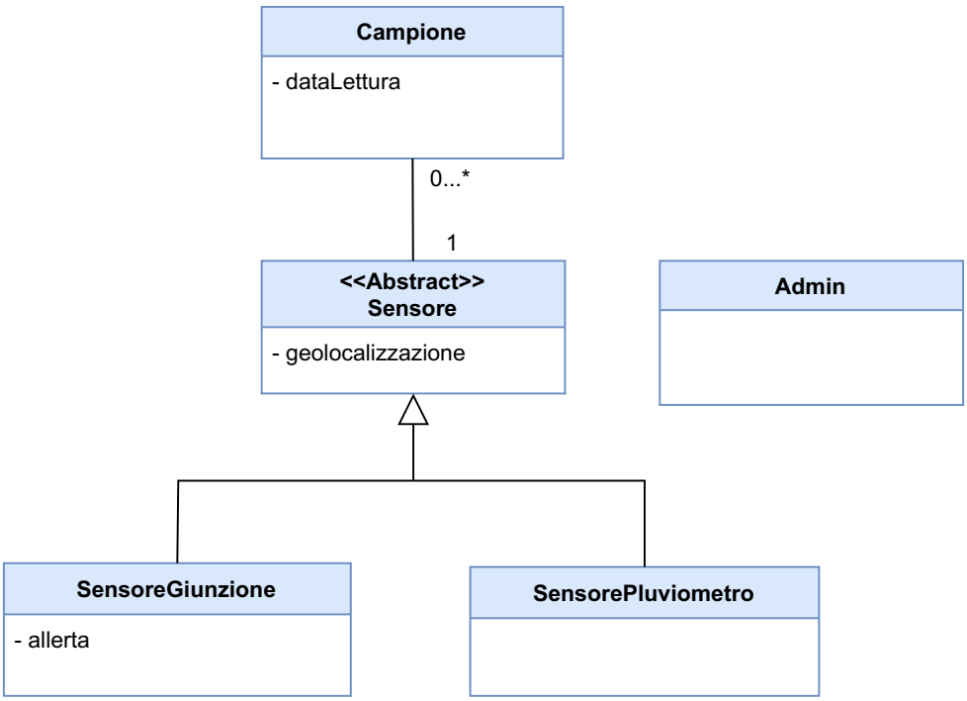


Figura 2: Diagramma di analisi delle classi

3.6 Modellazione Dati

All'interno del document database sono state individuate due principali collections: *userData* e *drainageData*

3.6.1 *userData* collection

Di seguito viene mostrato un esempio di documento relativo ad un utente:

```
{
        _id : <ObjectId>,
        ruolo : "admin",
        username : "admin1234",
        password : "a9'JF?Sk",
        anagrafica : {
                        nome : "Mario",
                        cognome : "Rossi",
                        email : "m.rossi@example.it"
                      }
}
```

In particolare, il campo **ruolo** potrebbe assumere valori come *admin, tecnico o visualizzatore.*

Di seguito viene mostrato un esempio di documento relativo ai parametri di configurazione dell'applicazione:

```
{
        _id: <ObjectId>,
        dataUltimaModifica : 2018-04-22T00:00:00.000+00:00,
        responsabileModifica : "admin1234",
        soglie : {
                sogliaAllertaRossa: 100,
                sogliaAllertaArancione: 55,
                sogliaAllertaGialla: 30
```

```
                }
}
```

Il campo **soglie** si riferisce ai livelli di acqua, espressi in mm, oltre i quali potrebbe verificarsi una situazione d'allerta.

3.6.2 *drainageData* collection

Esempio di un documento contenente le informazioni relative ad un sensore:

```
{
    _id: "TOS0100054408529",
    seriale: "TOS01000544",
    dataInstallazione: 2018-04-22T00:00:00.000+00:00,
    vecchiSeriali: [
                {
                    seriale:"TOS01000241",
                    dataInstallazione: 2016-12-07T00:00:00.000+00:00,
                    dataRimozione: 2017-07-11T00:00:00.000+00:00
                },
                {
                    seriale:"TOS01000253",
                    dataInstallazione: 2017-07-11T00:00:00.000+00:00,
                    dataRimozione: 2018-04-22T00:00:00.000+00:00
                }
            ],
    posizione: {
                latitudine: "142.2131413131",
                longitudine: "43.24324123123"
                },
    areaAmministrativa:{
                regione: "Toscana",
                provincia: "Pisa",
                citta: "Pisa",
                CAP: "56126",
                zonaDiRiferimento: "Dipartimento di Agraria",
                },
    tipo: "PLU",
    sottotipo: "Contatore"
    note: "Sensore va revisionato entro il prossimo anno"
```

```
}
```

In particolare:

- **_id** : codice seriale identificativo dell'area in cui è posto il sensore (corrisponde al campo IdDss della tabella *tagnamemaster* del database *adapt*)
- **dataInstallazione**: data in cui è stato installato il sensore
- **vecchiSeriali**: contiene il seriale del sensore rimosso, la data di installazione e la data in cui è stato rimosso (attualmente non implementato)
- **tipo**: possibili valori sono "PLU" per i pluviometri e "GIU" per le giunzioni
- **sottotipo**: possibili valori sono "Contatore" , "Portata Giornaliera", "Portata derivata da Contatore" (attualmente non implementato)
- **note**: possibilità di aggiungere note

Esempio di un documento contenente i campioni misurati da un sensore

```
{
    _id: <ObjectId>,
    idLuogo: "TOS0100054408529",
    dataRiferimento: 2019-12-03T00:00:00.000+00:00,
    timestampAggiornamento: 2019-12-04T18:36:14.000+00:00,
    attributoLIMS: null,
    campioni: [200.42,220.55, ... , 304.55],
    qualita: ["B","N", ... ,"I"],
    media: 242.59,
    messaggioErrore: "Errore...",
    simulato: false
}
```

In particolare:

- **idLuogo** : si riferisce all' _id del luogo in cui si trova il sensore

- **dataRiferimento** : corrisponde al campo DataLettura della tabella *tagnamemaster* del database *adapt*
- **timestampAggiornamento** : timestamp del momento effettivo in cui il dato giornaliero riferito a dataRiferimento è stato caricato nel database *adapt*
- **attributoLIMS** : utilizzato nelle letture LIMS (attualmente non implementato)
- **campioni** : array dei campioni raccolti durante la giornata, espressi in mm
- **qualità** : array delle qualità dei campioni raccolti. I possibili valori sono:
  - **B** = Buono
  - **I** = Interpolato
  - **N** = Non buono
  - **NULL** = Non inviato
- **media** : media calcolata dal sensore stesso, utilizzata come valore di confronto rispetto alla media calcolata considerando i valori ottenuti direttamente dai sensori. Il valore è espresso in mm
- **messaggioErrore** : eventuali messaggi d'errore da parte del sensore
- **simulato** : i possibili valori sono *true* e *false* (il valore *false* indica che i campioni provengono dai sensori)

Esempio di un documento contenente i campioni simulati ottenuti dal simulatore

```
{
    _id: <ObjectId>,
    idLuogo: "TOS0100054408529",
    dataRiferimento: 2019-12-03T00:00:00.000+00:00,
    campioni: [200.42,220.55, ... , 304.55],
    simulato: true
}
```

Il significato dei campi è analogo all'esempio precedente. Si noti che, in questo caso, il valore del campo **simulato** sarà sempre *true* perchè i campioni corrisponderanno all'output del simulatore.

4 Simulatore di campioni per le giunzioni

L'applicazione deve lavorare con dati reali importati dal database relazionale *adapt* e dati simulati generati dal simulatore SWMM5. Il problema nasce dal fatto che il simulatore SWMM5 permette di esportare i dati in un formato binario non importabile in Java, risulta quindi possibile importare esclusivamente i dati dal database relazionale il quale contiene solamente campioni reali relativi ai pluviometri.

Data la mancanza di campioni sulle giunzioni, reali e simulati, sarebbe impossibile testare l'applicazione quindi è stato necessario generare automaticamente i campioni mancanti.

4.1 Il Simulatore

Il simulatore **Generator** permette di generare campioni simulati e reali per una lista di sensori tra un certo intervallo temporale espresso in giorni, questi parametri vengono passati alla funzione.

```
Generator(MongoDBManager mongo, String[] sensorsId, String start,
↪  String end)
```

Il simulatore suddivide un giorno in quattro finestre temporali, per ogni finestra decide con una certa probabilità se sta piovendo oppure no:

- **se piove** il livello di acqua nella giunzione inizia ad aumentare
- **se non piove** inizia a diminuire simulando lo svuotamento della condotta

il livello di acqua viene mantenuto nel passaggio da una finestra temporale all'altra.

Sono presi in considerazione anche eventi di precipitazioni eccezionali, durante i quali l'incremento del livello d'acqua nella condotta è particolarmente significativo.

Vengono generati per ogni giorno **96 campioni** (uno ogni 15 minuti), quando uno di questi deve essere generato, si prende l'attuale valore della condotta e lo si incrementa o diminuisce, a seconda se sta o meno piovendo, di un valore randomico

scelto in un dato intervallo. Una volta ottenuti i campioni di un giorno, si applica del rumore per generare i valori reali per tutti i sensori, questo viene fatto perchè, tenendo conto che i sensori appartengono alla stessa città, le precipitazioni saranno pressochè simili ma non del tutto uguali.

Una volta generati i campioni reali, applicando rumore su ognuno di essi si generano i campioni simulati in modo che ci sia una variazione rispetto a quelli reali.

Ad ogni campione reale viene anche associata una qualità come da specifiche (N, I o B), anche questa è generata in maniera pseudocasuale rendendo preponderante la presenza di dati buoni B (94%). Oltre a questo viene anche generato, in rari casi, un messaggio di errore associato alla trasmissione dei dati.

All'interno della classe è possibile modificare alcuni parametri del simulatore:

- **livelloAttuale:** livello iniziale di acqua nella condotta, in *mm*
- **variazioneLivello:** variazione, positiva o negativa, del livello di acqua nella condotta alla generazione di un campione, in *mm*
- **campioniGiornalieri:** numero di campioni giornalieri che devono essere generati
- **livelloMassimo:** massimo livello di acqua nella condotta, in *mm*
- **variazioneEventoAltaIntensita:** variazione acqua nella condotta durante un evento di pioggia ad alta intensità, il valore viene sommato a quello di *variazioneLivello*, in *mm*
- **probabilitaPioggia:** probabilità che in una finestra temporale ci sia una precipitazione
- **probabilitaEventoAltaIntensitaSePiove:** probabilità che in una finestra temporale dove c'è una precipitazione ci sia un evento ad alta intensità
- **variazioneRumore:** valore utilizzato per generare l'intervallo del rumore, in *mm*

La classe è inserita in un package differente da quello dell'applicazione visto che potrà essere rimossa quando sarà disponibile un flusso completo di dati. I dati vengono inseriti su MongoDB sfruttando i metodi nella classe *MongoDBManager*.

5 Implementazione del database MongoDB

5.1 Descrizione delle classi

È possibile suddividere le varie classi in cinque categorie, in base al loro ruolo all'interno dell'applicazione, ovvero:

- **Visualizzazione grafica**
- **Rappresentazione degli oggetti**
- **Interfaccia verso server MySQL**
- **Interfaccia verso server MongoDB**
- **Configurazione**

5.1.1 Visualizzazione grafica

All'interno di questa sezione troviamo tutte le classi che offrono una vista all'utente dell'applicazione:

- **LoginSceneController:** presenta la scena relativa alla visualizzazione della pagina di login, in cui l'utente può scegliere se entrare come ospite, e avere la possibilità solo di visualizzare i dati forniti dall'applicazione, oppure se autenticarsi, e avere anche la possibilità di modificare i dati delle giunzioni o pluviometri, le soglie o aggiungere nuovi utenti.
- **AdminSceneController:** presenta la scena relativa alla visualizzazione della pagina privata dell'amministratore, dove è possibile modificare le soglie di allarme e i propri dati personali, oppure importare, tramite file txt o json, dei nuovi sample.
- **JunctionSceneController:** presenta la scena relativa alla visualizzazione della descrizione di una determinata giunzione, dando all'utente la possibilità di vedere grafici usando dati reali e/o simulati per qualsiasi giorno, mese o anno. Fornisce inoltre la possibilità di mostrare statistiche aggregate per i sopracitati periodi.

- **RainGaugeSceneController:** presenta la scena relativa alla visualizzazione della descrizione di un determinato pluviometro, dando all'utente la possibilità di vedere grafici usando dati reali per qualsiasi giorno, mese o anno. Fornisce inoltre la possibilità di mostrare statistiche aggregate per i sopracitati periodi.
- **ListJunctionSceneController:** presenta la scena relativa alla visualizzazione della lista di tutte le giunzioni, fornendo all'utente la possibilità di interagire con ognuna di esse.
- **ListRainGaugeSceneController:** presenta la scena relativa alla visualizzazione della lista di tutti i pluviometri, fornendo all'utente la possibilità di interagire con ognuno di essi.
- **MapSceneController:** presenta la scena relativa alla visualizzazione della mappa, mostrando su richiesta dell'utente valori provenienti da statistiche aggregate per sensori di una stessa città su un dato arco temporale.
- **NewsSceneController:** presenta la scena relativa alla visualizzazione delle news, mostrando per la città selezionata di default, quali giunzioni sono in allerta nei prossimi due giorni in base alle previsioni fornite dal simulatore, dando all'utente la possibilità di interagire con essi. Viene inoltre mostrata una tabella in cui per ogni sensore viene mostrato da quanti giorni non fornisce al database MongoDB dati aggiornati.
- **SensorTableView:** contiene i dati utilizzati per popolare la tabella nella scena delle news.

5.1.2 Rappresentazione degli oggetti

All'interno di questa sezione fanno parte tutte le classe che rappresentano oggetti all'interno dell'applicazione:

- **Admin:** Rappresenta l'utente admin e serve per controllare se è possibile modificare dati all'interno dell'applicazione ed eventualmente tenerne traccia.
- **Sensor:** Rappresenta un sensore generico, fornendo costruttori e funzioni get e set per ogni membro.

- **JunctionSensor:** Estende la classe Sensor e rappresenta una giunzione.
- **RainGaugeSensor:** Estende la classe Sensor e rappresenta un pluviometro.
- **Sample:** Rappresenta un sample (campione) generico, fornendo costruttori e funzioni get e set per ogni membro.
- **SimulatedSample:** Estende la classe Sample e rappresenta un sample simulato (cioè fornito dal simulatore).
- **MeasuredSample:** Estende la classe Sample e rappresenta un sample reale (cioè proveniente dai sensori).
- **MapManager:** Rappresenta l'oggetto mappa con relativi metodi per fare zoom in e zoom out, creare e gestire i cerchi sopra essa che rappresentano i sensori.

5.1.3 Interfaccia verso server MySQL

Questa sezione è composta da tutte le classi che consentono la retrocompatibilità con il database MySQL e permettono l'import sul database MongoDB di vari sample e sensori.

- **SQLDBManager:** Questa classe permette l'interazione tra l'applicazione Java e il database MySQL, permettendo di ottenere informazioni tramite diverse query.
- **SampleJPA:** Serve per definire gli oggetti sample tramite JPA e rappresenta una parte dei campi della tabella *dssdati* del database MySQL.
- **SampleKeyJPA:** Rappresenta la chiave composta della tabella *dssdati* all'interno del database MySQL.
- **TagnameJPA:** Serve per definire gli oggetti sensore tramite JPA e rappresenta una parte dei campi della tabella *tagnamemaster* del database MySQL.
- **TagnameKeyJPA:** Rappresenta la chiave composta della tabella *tagnamemaster* all'interno del database MySQL.

5.1.4 Interfaccia verso server MongoDB

MongoDBManager è la classe fondamentale dell'applicazione, in quanto fornisce la connessione con il server MongoDB. Implementa metodi che consentono qualsiasi operazione CRUD, tra cui aggregazioni, import di dati, query per la visualizzazioni di dati etc.

5.1.5 Configurazione

La classe **ConfigParameter** serve per importare i parametri di configurazione provenienti da un file esterno. In esso saranno presenti informazioni riguardo l'indirizzo del server, utente e password per l'autenticazione e la città di default a cui l'applicazione deve far riferimento.

5.2 Pipeline di aggregazione

Uno degli scopi principali dell'applicazione è quello di mostrare statistiche relative ai dati presenti nel database. Per questo motivo in questa sezione sono riportate le varie pipeline di aggregazione create in MongoDB.

5.2.1 Media, minimo e massimo in un dato intervallo temporale

Il primo tipo di pipeline di aggregazione utilizzata è quello relativo alla media, massimo o minimo valore di un dato intervallo temporale, di cui è riportato un esempio:

```
MongoCursor<Document> cursor =
↪   coll.get("drainageData").aggregate(Arrays.asList(
    Aggregates.unwind("$campioni"),
    Aggregates.match(and(eq("idLuogo",idSensor),
    gte("dataRiferimento",start),
    lte("dataRiferimento",end),eq("simulato",simulato))),
    Aggregates.group("$idLuogo",Accumulators.avg("avg","$campioni"))
)).cursor();
```

Prima di tutto viene utilizzato l'operatore **unwind**, necessario per dividere l'array di double contenente i valori misurati o simulati riferiti ad una certa data in un gruppo di documenti distinti, utilizzabili in seguito dagli altri operatori di aggregazione.
Dopo questo si trova il passo relativo al **match**, ovvero si selezionano solo quei campioni relativi ad uno specifico sensore, identificato dal suo id (*idSensor*), compresi tra le date *start* ed *end* e che siano o meno valori misurati o simulati, a seconda del valore della variabile *simulato*.
Infine viene utilizzato l'operatore **group** per raggruppare tutti i valori dei campioni nell'intervallo temporale selezionato e calcolarne la media. Come *_id* viene utilizzato il campo *idLuogo*, ovvero l'identificativo del sensore per un certo array di campioni.

5.2.2 Media, minimo e massimo di un dato mese dell'anno

Il secondo tipo di pipeline di aggregazione utilizzata è quello relativo alla media, massimo o minimo valore giorno per giorno di un dato mese dell'anno, di cui è riportato un esempio:

```
MongoCursor<Document> cursor =
↪  coll.get("drainageData").aggregate(Arrays.asList(
    Aggregates.project(Projections.fields(
        Projections.computed("day", new
        ↪  Document("$dayOfMonth","$dataRiferimento")),
        Projections.computed("resMonth", new
        ↪  Document("$month","$dataRiferimento")),
        Projections.computed("resYear", new
        ↪  Document("$year","$dataRiferimento")),
        Projections.include("campioni"),
        Projections.include("simulato"),
        Projections.include("idLuogo")
    )),
    Aggregates.unwind("$campioni"),
    Aggregates.match(and(eq("resYear",year),eq("resMonth",month),
    eq("idLuogo", idSensor),eq("simulato",simulato))),
```

```
    Aggregates.group("$day",Accumulators.avg("avg","$campioni"))
)).cursor();
```

Prima di tutto viene utilizzato l'operatore **project** per proiettare i campi necessari per l'aggregazione (*campioni*, *simulato* e *idLuogo*) e per estrarre dalla *dataRiferimento* il giorno, il mese e l'anno corrispondente.
Dopo di che, come nel tipo precedente di aggregazione, viene applicato l'operatore **unwind** per dividere l'array di double contenente i valori misurati o simulati riferiti ad una certa data in un gruppo di documenti distinti, utilizzabili in seguito dagli altri operatori di aggregazione.
Dopo questo si trova il passo relativo al **match**, ovvero si selezionano solo quei campioni relativi ad uno specifico sensore, identificato dal suo id (*idSensor*), con l'anno ed il mese specificato dalle variabili ottenute al passo precedente e che siano o meno valori misurati o simulati, a seconda del valore della variabile *simulato*.
Infine viene utilizzato l'operatore **group** per raggruppare tutti i valori dei campioni selezionati e calcolarne la media. Come *_id* viene utilizzato il campo *day*, ovvero il numero che identifica il giorno all'interno del mese.

5.2.3 Media, minimo e massimo di un dato anno

Il terzo tipo di pipeline di aggregazione utilizzata è quello relativo alla media, massimo o minimo valore mese per mese di un dato anno, di cui è riportato un esempio:

```
MongoCursor<Document> cursor =
↪  coll.get("drainageData").aggregate(Arrays.asList(
       Aggregates.project(Projections.fields(
               Projections.computed("resMonth", new
               ↪  Document("$month","$dataRiferimento")),
               Projections.computed("resYear", new
               ↪  Document("$year","$dataRiferimento")),
               Projections.include("campioni"),
               Projections.include("simulato"),
               Projections.include("idLuogo")
```

```
        )),
        Aggregates.unwind("$campioni"),
        Aggregates.match(and(eq("resYear",year),eq("idLuogo",
        ↪  idSensor),
        eq("simulato",simulato))),
        Aggregates.group("$resMonth",Accumulators.avg("avg","$campioni"))
)).cursor();
```

Prima di tutto viene utilizzato l'operatore **project** per proiettare i campi necessari per l'aggregazione (*campioni*, *simulato* e *idLuogo*) e per estrarre dalla *dataRiferimento* il mese e l'anno corrispondente.
Dopo di che, come nel tipo precedente di aggregazione, viene applicato l'operatore **unwind** per dividere l'array di double contenente i valori misurati o simulati riferiti ad una certa data in un gruppo di documenti distinti, utilizzabili in seguito dagli altri operatori di aggregazione.
Dopo questo si trova il passo relativo al **match**, ovvero si selezionano solo quei campioni relativi ad uno specifico sensore, identificato dal suo id (*idSensor*), con l'anno specificato dalle variabili ottenute al passo precedente e che siano o meno valori misurati o simulati, a seconda del valore della variabile *simulato*.
Infine viene utilizzato l'operatore **group** per raggruppare tutti i valori dei campioni selezionati e calcolarne la media. Come *_id* viene utilizzato il campo *resMonth*, ovvero il numero che identifica il mese all'interno dell'anno.

5.3 Indici MongoDB

L'applicazione dovrà essere predisposta per gestire un gran quantitativo di dati, i quali dovranno essere acceduti frequentemente ai fini di visualizzare informazioni sui sensori, statistiche e previsioni sui loro dati.
Garantire queste operazioni gestendo una tale mole di dati senza usare ottimizzazioni renderebbe l'applicazione poco reattiva, introducendo notevoli latenze.
Per questo motivo è stata svolta un'analisi relativa agli accessi al database in modo da poter introdurre indici rendendo l'applicazione più reattiva alle richieste dell'utente.

In questo paragrafo verranno mostrati i vari indici aggiunti, motivandone la loro utilità, e mostrando le statistiche in termini di performance che essi comportano.

5.3.1 Indice sui campi idLuogo, dataRiferimento

La maggior parte delle operazioni previste dall'applicazione coinvolgono query di ricerca sul database MongoDB per ottenere un insieme di campioni (reali e non) in un determinato arco temporale da un sensore prestabilito. Utilizzando questo indice, la ricerca dei dati visualizzati nella homepage dell'applicazione, che permette di visualizzare i sensori in stato di allerta, è immediata.

Per la creazione dell'indice è stato usato il seguente comando:

```
db.drainageData.createIndex({idLuogo: 1, dataRiferimento: -1} )
```

Il primo attributo ordina in modo crescente secondo la posizione del sensore, mentre il secondo ordina in modo decrescente secondo la data di riferimento dei campioni. Questo è utile perchè campioni relativi a date recenti nell'applicazione vengono utilizzati più di frequente e consecutivamente.

Nella seguente immagine vengono mostrate le performance con e senza indici relative alla seguente query:

```
db.drainageData.find({$and:[{idLuogo:{$eq: "FGN_PI_10"}},
↪  {dataRiferimento:{$lte:ISODate("2016-01-09T00:00:00Z")}},
↪  {dataRiferimento:{$gte:ISODate("2016-01-06T00:00:00Z")}}]})
.explain("executionStats")
```

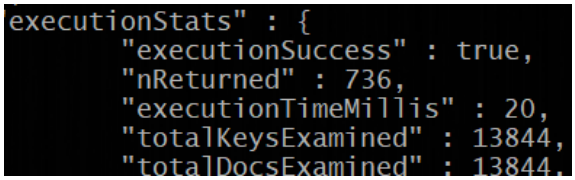


Figura 3: Performance su ricerca senza indicizzazione

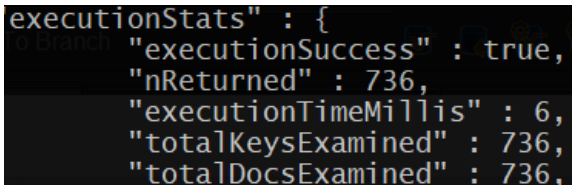


Figura 4: Performance su ricerca con indicizzazione

Una miglior visualizzazione grafica di questo risultato è visibile attraverso il confronto dei due risultati tramite istogramma presente nella seguente immagine:

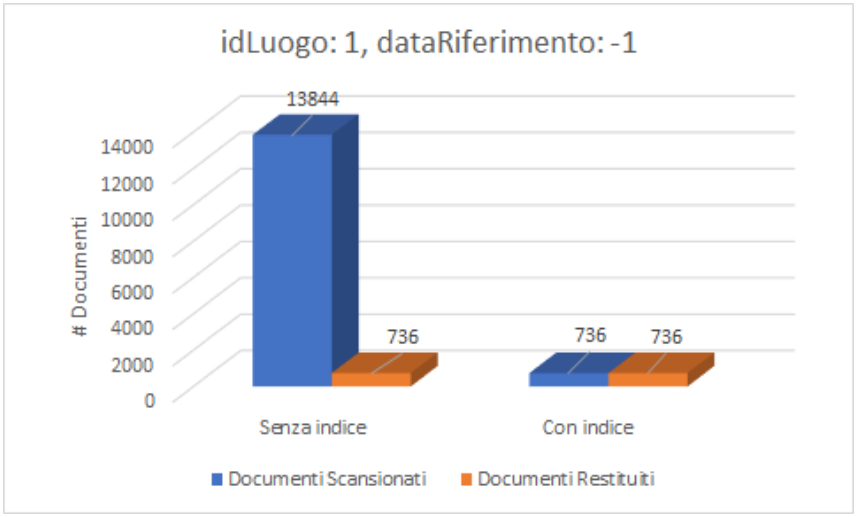


Figura 5: Confronto delle performance con e senza uso di indici

5.3.2 Indice sui campi idLuogo, timestampAggiornamento

Questo indice è necessario per ottimizzare l'ultima trasmissione di dati effettuata da un sensore, in modo da poterlo sfruttare per capire da quanto tempo non trasmette e per sapere quali dati, non ancora presenti sul database MongoDB, è necessario caricare dal database relazionale.

Per la creazione dell'indice è stato usato il seguente comando:

```
db.drainageData.createIndex({ idLuogo:1, timestampAggiornamento:-1 })
```

Il primo attributo ordina in modo crescente secondo la posizione del sensore, mentre il secondo ordina in modo decrescente secondo il timestamp di aggiornamento dei campioni.

Nella seguente immagine vengono mostrate le performance con e senza indici relative alla seguente query:

```
db.drainageData.find({$and:[{idLuogo:{$eq:"FGN_PI_10"}},
↪  {simulato:{$eq: false}}]})
.sort({timestampAggiornamento:-1})
.limit(1)
.explain("executionStats")
```

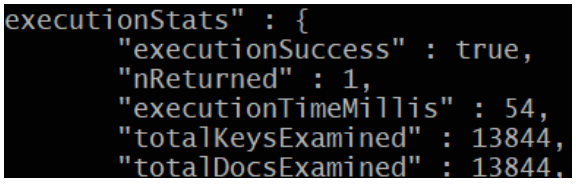


Figura 6: Performance su ricerca senza indicizzazione

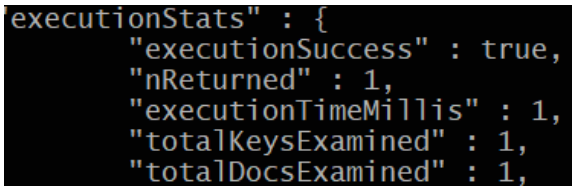


Figura 7: Performance su ricerca con indicizzazione

Una miglior visualizzazione grafica di questo risultato è visibile attraverso il confronto dei due risultati tramite istogramma presente nella seguente immagine:

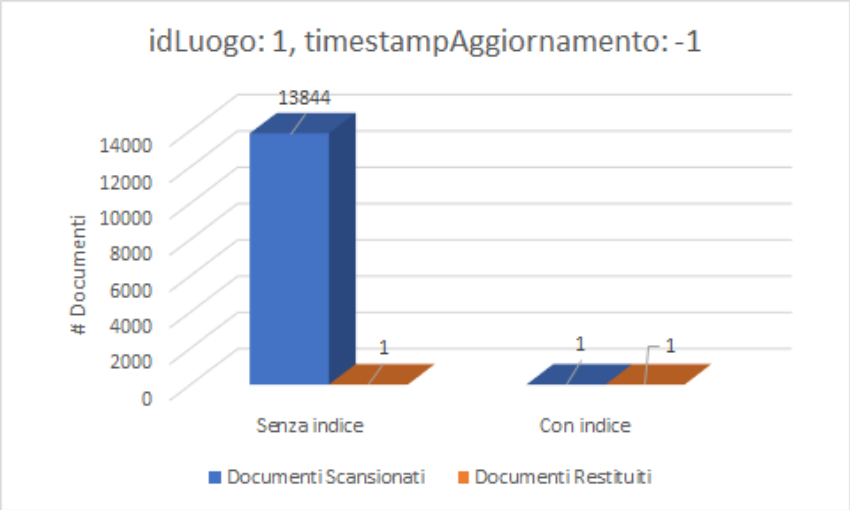


Figura 8: Confronto delle performance con e senza uso di indici

5.3.3 Indice sul campo citta

Questo indice è necessario per ottimizzare la ricerca dei sensori appartenenti ad una specifica città e delle loro relative statistiche.

Per la creazione dell'indice è stato usato il seguente comando:

```
db.drainageData.createIndex({"areaAmministrativa.citta": 1})
```

L'attributo ordina in ordine crescente in base alla città riferita, in modo che dati relativi alla stessa città e quindi visualizzati nello stesso momento si trovino in sequenza.

Nella seguente immagine vengono mostrate le performance con e senza indici relative alla seguente query:

```
db.drainageData.find({"areaAmministrativa.citta":{$eq:"Pisa"}})
.explain("executionStats")
```

```
executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 6,
        "executionTimeMillis" : 70,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 84771,
```


Figura 9: Performance su ricerca senza indicizzazione


```
executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 6,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 6,
        "totalDocsExamined" : 6,
```


Figura 10: Performance su ricerca con indicizzazione

Una miglior visualizzazione grafica di questo risultato è visibile attraverso il confronto dei due risultati tramite istogramma presente nella seguente immagine:

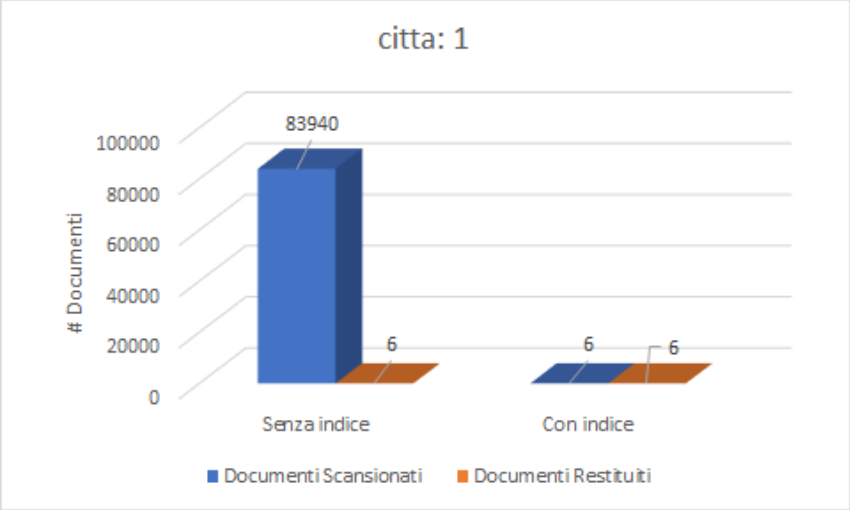


Figura 11: Confronto delle performance con e senza uso di indici

5.4 Replicazione del database

Il database è stato sviluppato su un cluster remoto di server. In particolare, per implementare la replicazione sono stati utilizzati i server corrispondenti agli indirizzi IP mostrati in figura.

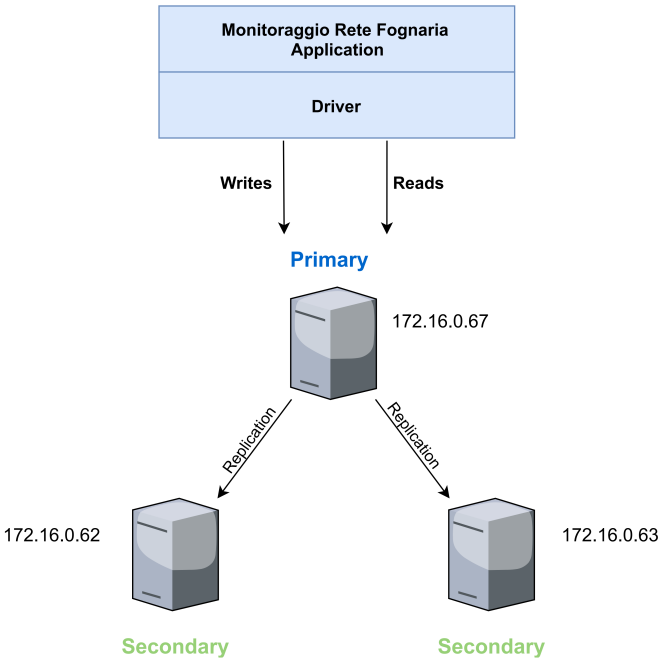


Figura 12: Architettura del cluster

Il *replica set* è costituito da un unico **primary** server in grado di accettare le letture e le scritture. I **secondary** server replicano l'*oplog* del primario e, nel caso in cui quest'ultimo non fosse disponibile, effettuano un'elezione nominando un nuovo primary server tra quelli disponibili.
A tal proposito, sono stati effettuati dei test disconnettendo alcuni dei server del cluster e verificando l'elezione di un nuovo server primario e il corretto funzionamento dell'applicazione.

5.5 Sharded cluster

Una possibile futura feature sarebbe la trasformazione del database da Replica set a sharded cluster, come mostrato in figura seguente:

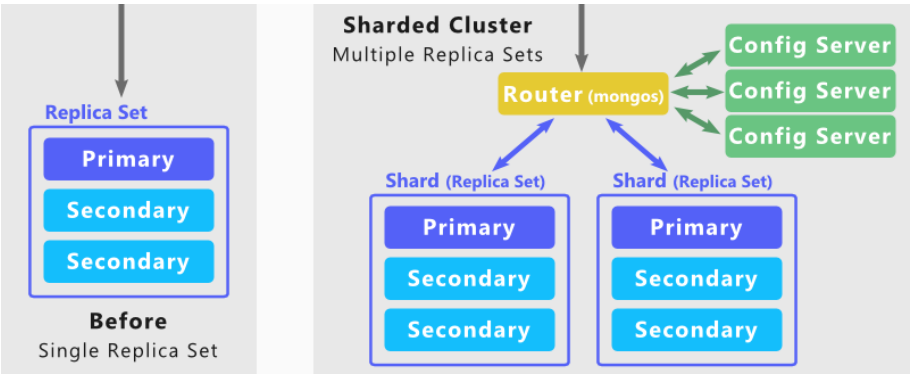


Figura 13: Architettura dello sharded cluster

Vista la grossa mole di dati da memorizzare e gestire, sarebbe opportuno dividere il database in più shard.
Possono essere adottate soluzioni diverse nella scelta della shard key:

- **Shard key su dataRiferimento:** Si potrebbe scegliere come shard key la *dataRiferimento*, in questo modo gli shard con relative repliche andranno a memorizzare campioni di diversi sensori suddivisi per data, separando i dati più recenti da quelli meno recenti.

- **Shard key su città:** Nel caso in cui i server del database fossero collocati in diverse aree geografiche, ad esempio nei pressi delle città in cui si trovano i sensori, allora sarebbe opportuno usare come shard key il campo *città*. In questo modo ogni shard con relative repliche andrà a memorizzare tutti e soli i campioni, con qualsiasi data di riferimento, appartenti a sensori della stessa città.

6 Applicazione lato Front-End

Nel seguito descriveremo l'applicazione lato Front-End, con relative interfacce grafiche.

6.1 Login

L'accesso all'applicazione è consentito in due diverse modalità: con e senza autenticazione.

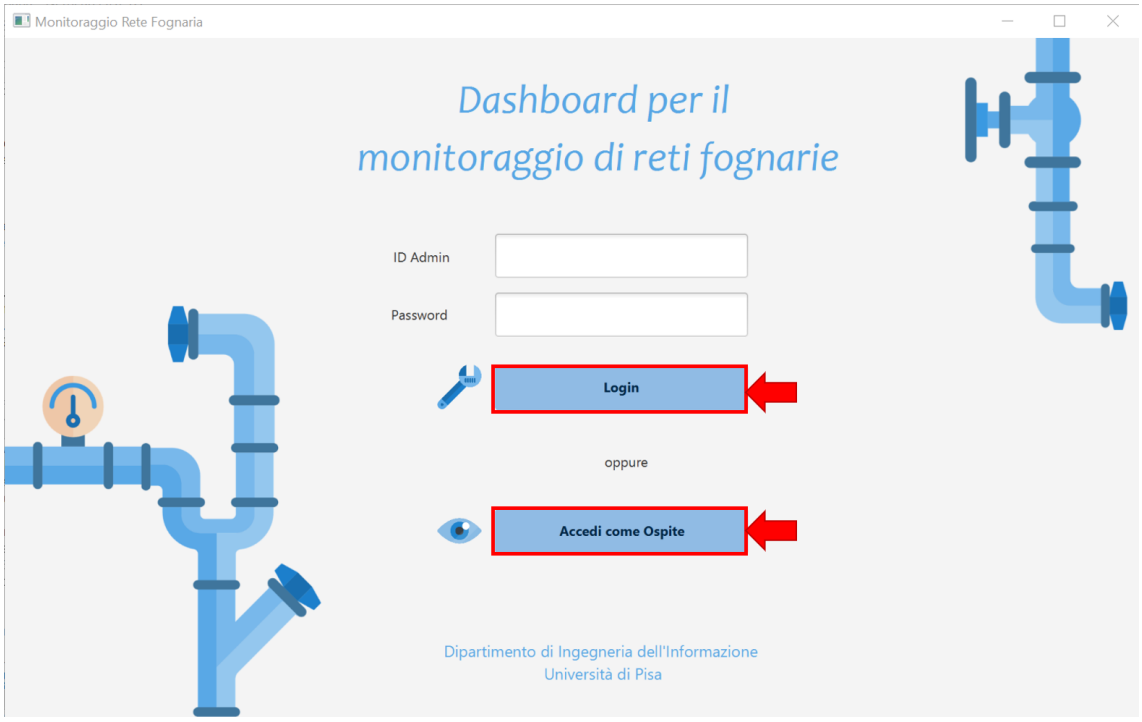


Figura 14: Accesso all'applicazione

L'autenticazione è riservata solamente all'amministratore del sistema che, dopo aver inserito username e password e aver premuto il tasto **Login**, potrà visualizzare e modificare le informazioni presenti (esiste un utente di default con username *Admin* e password *admin*). Un utente generico, invece, potrà accedere all'applicazione premendo il tasto **Accedi come Ospite** e sarà in grado di visualizzare le informazioni,

ma non di modificarle. Inoltre, un utente generico non avrà accesso all'Area Privata dell'applicazione.

6.2 News

La mappa mostra il livello di allerta previsto sulle giunzioni per le successive 48 ore. Cliccando con il tasto sinistro del mouse sul simbolo colorato è possibile zoomare sull'area adiacente, cliccando, invece, con il tasto destro l'utente visualizzerà i grafici relativi alla giunzione selezionata.
I grafici relativi ad un sensore possono essere visualizzati anche selezionando la riga della tabella corrispondente al sensore stesso. Cliccando, invece, su **Aggiorna sensori** è possibile aggiornare lo stato dei sensori all'interno dell'interfaccia grafica dell'applicazione.

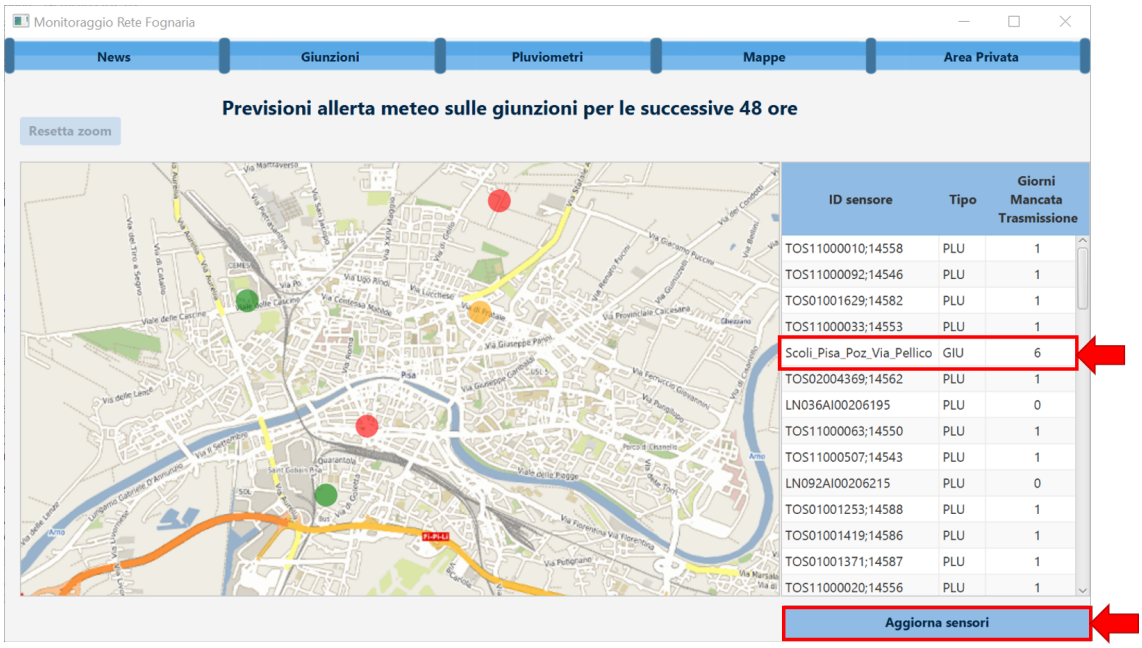


Figura 15: Mappa contenente il livello di allerta delle giunzioni

6.3 Elenco Giunzioni

La tabella mostra le informazioni più importanti delle giunzioni presenti all'interno del database. Cliccando su una riga della tabella è possibile visualizzare le informazioni relative alla giunzione selezionata. Tutte le informazioni possono essere modificate dall'amministratore e salvate premendo il tasto **Salva**, ad eccezione di *ID Sensore*, *Seriale* e *Data Installazione* che sono campi non editabili.

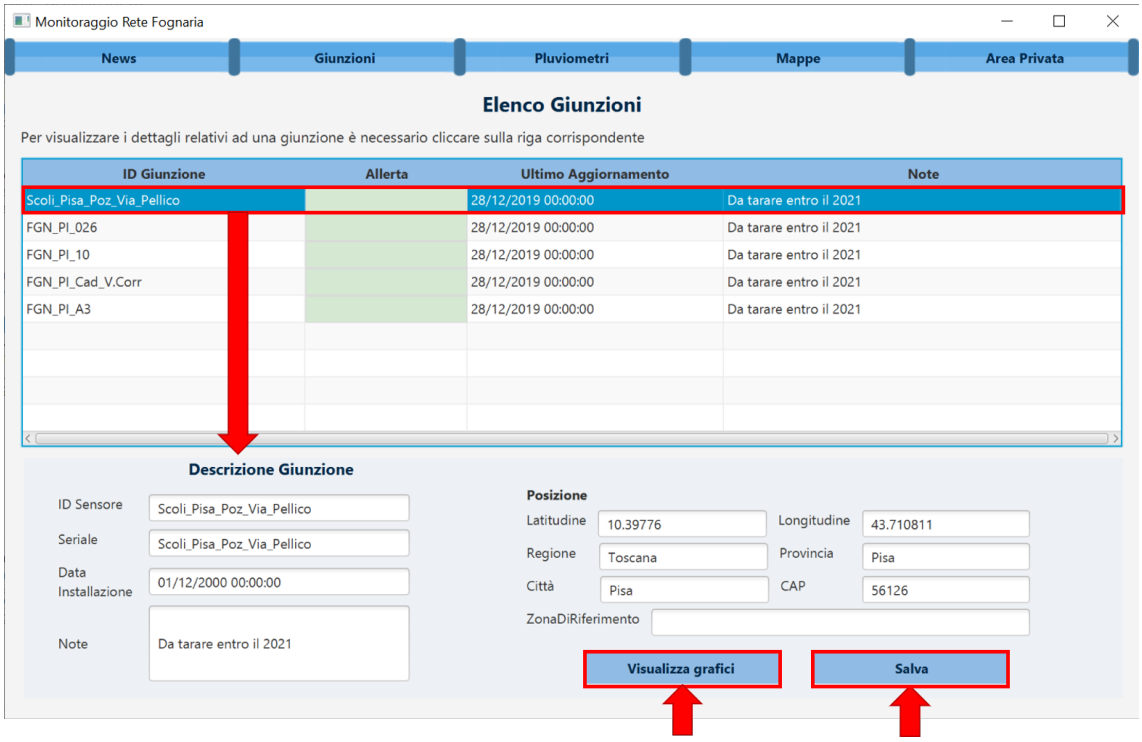


Figura 16: Elenco Giunzioni

Cliccando su **Visualizza grafici** sarà possibile visualizzare graficamente i dati relativi alla giunzione selezionata.

6.4 Visualizzazione dati giunzione

Il grafico presente all'interno della schermata mostra il livello d'acqua presente all'interno della giunzione selezionata. E' possibile visualizzare i dati scegliendo quattro

diverse granularità: dati recenti, dati storici giornalieri, dati storici mensili e dati storici annuali.

6.4.1 Dati recenti

In questa schermata (che è quella mostrata di default) è possibile visualizzare i dati effettivamente misurati dal sensore, relativi al giorno precedente rispetto a quello attuale (in rosso) ed i dati simulati del giorno corrente e del giorno successivo (in giallo). É possibile accedere nuovamente a questa visualizzazione cliccando su **Visualizza i dati misurati nelle 24 ore precedenti e i dati previsti per le 48 ore successive rispetto alla data odierna**.

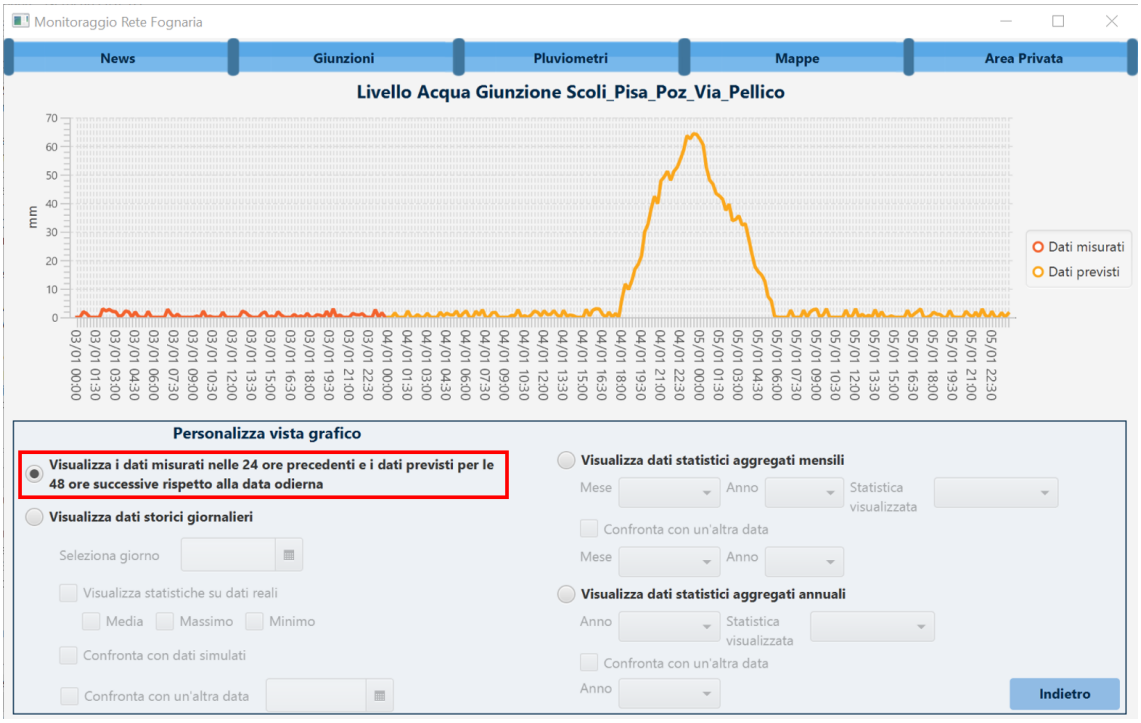


Figura 17: Livello d'acqua misurato nelle 24 ore precedenti e previsto nelle 48 ore successive rispetto alla data odierna

6.4.2 Dati storici giornalieri

Tramite questa sezione è possibile visualizzare dati storici misurati relativi ad una data selezionata. Per visualizzare i dati è necessario cliccare su **Visualizza dati storici giornalieri** e successivamente selezionare una data dal calendario presente a destra di *Seleziona giorno*. Dopo di che sono disponibili tre ulteriori opzioni:

- Visualizzazione di statistiche sui dati del giorno selezionato: selezionare **Visualizza statistiche su dati reali** e successivamente barrare le caselle **Media**, **Massimo** e **Minimo** per visualizzare le relative statistiche

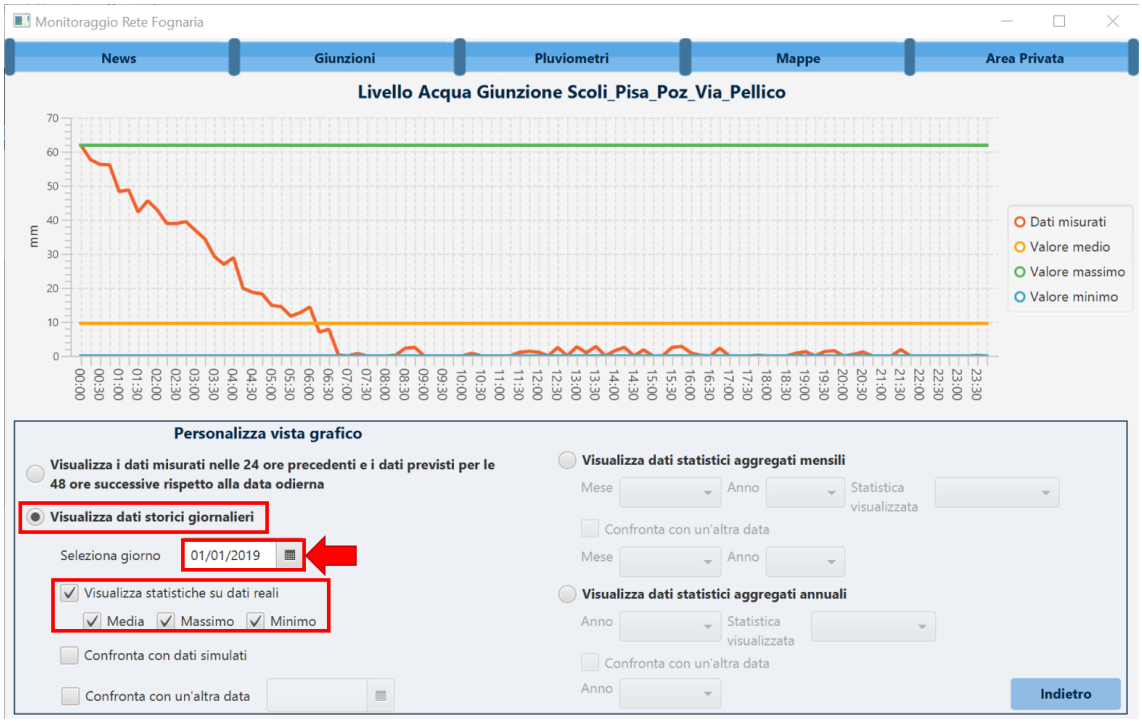


Figura 18: Dati storici giornalieri con statistiche

- Confronto con i dati simulati: selezionare **Confronta con i dati simulati** per visualizzare la serie di dati simulati relativa allo stesso giorno selezionato
- Confronto con i dati misurati di un'altra data: selezionare **Confronta con un'altra data** e successivamente selezionare una data nel calendario adiacente

per visualizzare nel grafico i dati relativi ad un'altra data da confrontare (per leggibilità se, prima della selezione della data di confronto, sul grafico erano presenti dati simulati, questi ultimi vengono rimossi)

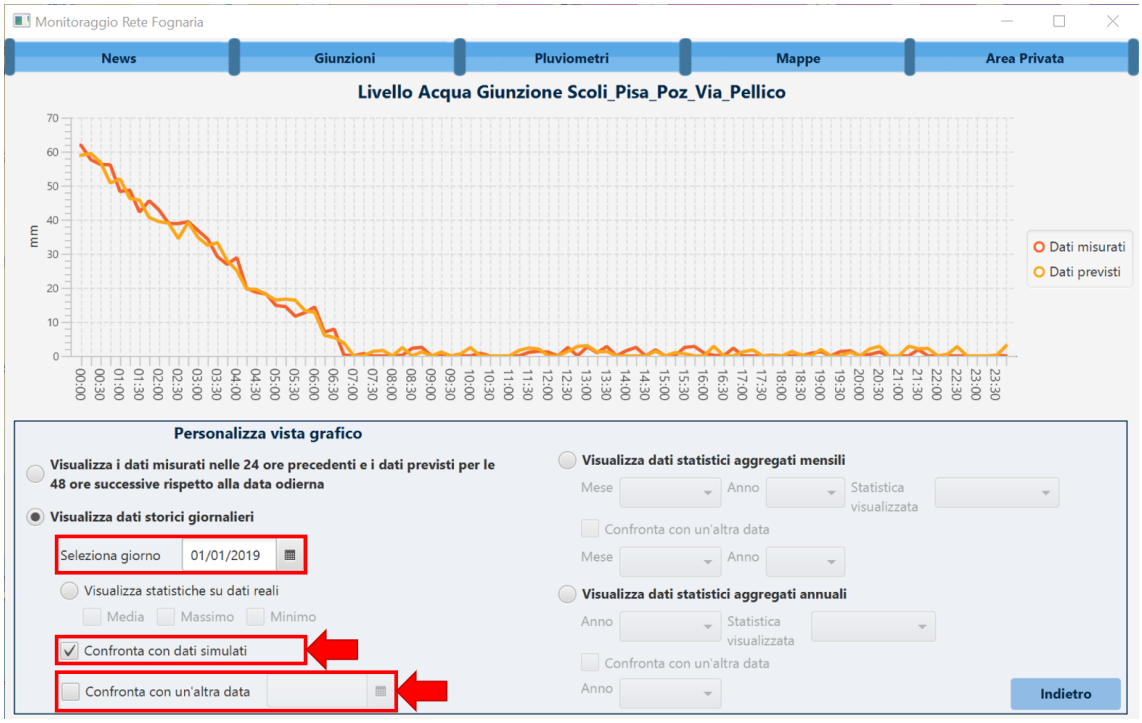


Figura 19: Dati storici giornalieri e confronto con dati simulati

6.4.3 Dati storici mensili

Per visualizzare dati aggregati su base giornaliera di una specifica coppia mese/anno è necessario selezionare **Visualizza dati statistici aggregati mensili**, selezionare **Mese** ed **Anno** desiderati e selezionare la statistica desiderata dal menù a tendina a scelta tra **Media**, **Massimo** o **Minimo**.
Se si desidera fare un confronto con un altro mese/anno è necessario selezionare **Confronta con un'altra data** e successivamente selezionare **Mese** ed **Anno** desiderati dai menù a tendina.

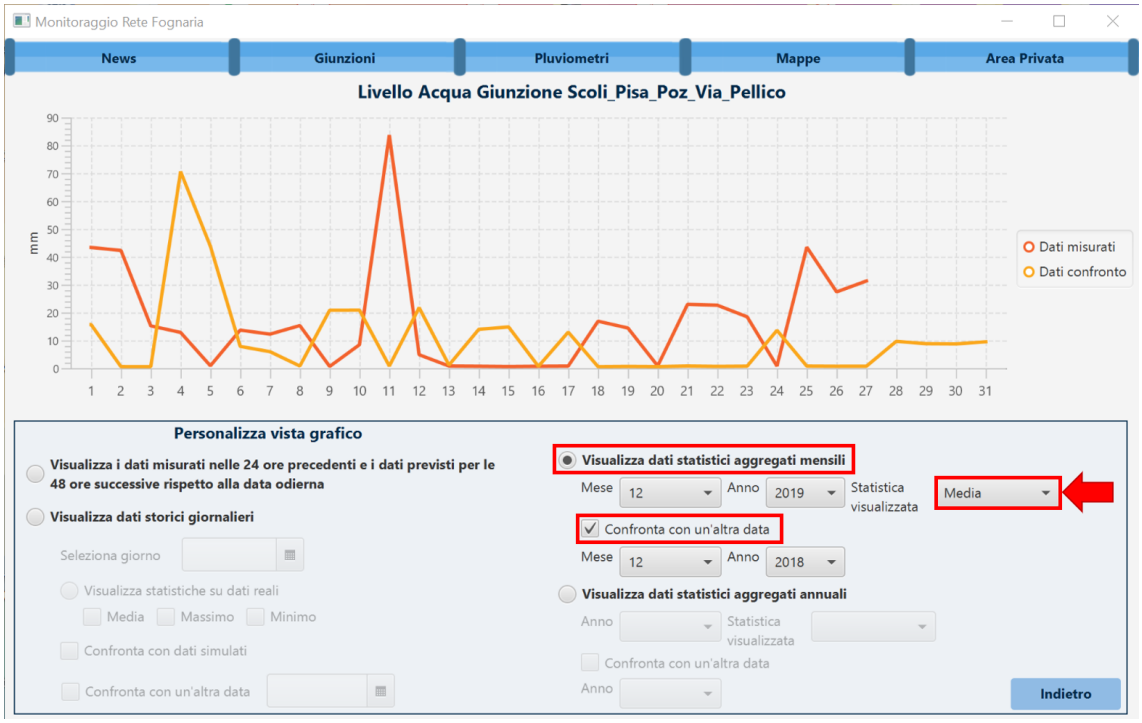


Figura 20: Dati storici mensili e confronto con un'altra data

6.4.4 Dati storici annuali

Per visualizzare dati aggregati su base mensile di uno specifico anno è necessario selezionare **Visualizza dati statistici aggregati annuali**, selezionare **Anno** desiderato e selezionare la statistica desiderata dal menù a tendina a scelta tra **Media**, **Massimo** o **Minimo**.
Se si desidera fare un confronto con un altro anno è necessario selezionare **Confronta con un'altra data** e successivamente selezionare **Anno** desiderato dal menù a tendina.

Infine è possibile cliccare su bottone **Indietro** per tornare alla schermata *Giunzioni* dove è presente l'elenco delle giunzioni disponibili con i dettagli della giunzione selezionata precedentemente.

Figura 21: Dati storici annuali e confronto con un'altra data

6.5 Pluviometri

La sezione dedicata ai pluviometri offre le stesse funzionalità di quella dedicata alle giunzioni ad eccezione della visualizzazione dei dati simulati e dell'allerta prevista. Questo perché, per i pluviometri, non si hanno a disposizione i dati simulati ma solamente quelli effettivamente misurati dai sensori.

L'interfaccia offre la visualizzazione congiunta piogge/livelli e piogge/portate (ovvero ietogramma/idrogramma): questo consente di poter correlare più direttamente afflussi e deflussi. Riguardo al plot dei dati di pioggia occorre tener presente che la granularità temporale può essere disomogenea. Ad esempio, i dati inviati da SIR sono ogni 15 minuti, mentre la quantità di pioggia prevista da LAMMA è di ora in ora.

6.6 Mappe e dati statistici

Tramite il menù posto sulla destra, è possibile impostare i parametri sfruttati dalle statistiche. In particolare, è possibile scegliere la città, la vista (IdSensore, Media, Massimo, Minimo) e l'intervallo di tempo su cui effettuare la statistica. Dopo aver selezionato tutti i parametri, bisogna premere il tasto **Genera statistiche**.
Nei tre campi sottostanti verranno visualizzati i risultati ottenuti.
Tramite il tasto **Aggiorna sensori** è possibile aggiornare lo stato dei sensori all'interno dell'interfaccia grafica dell'applicazione.

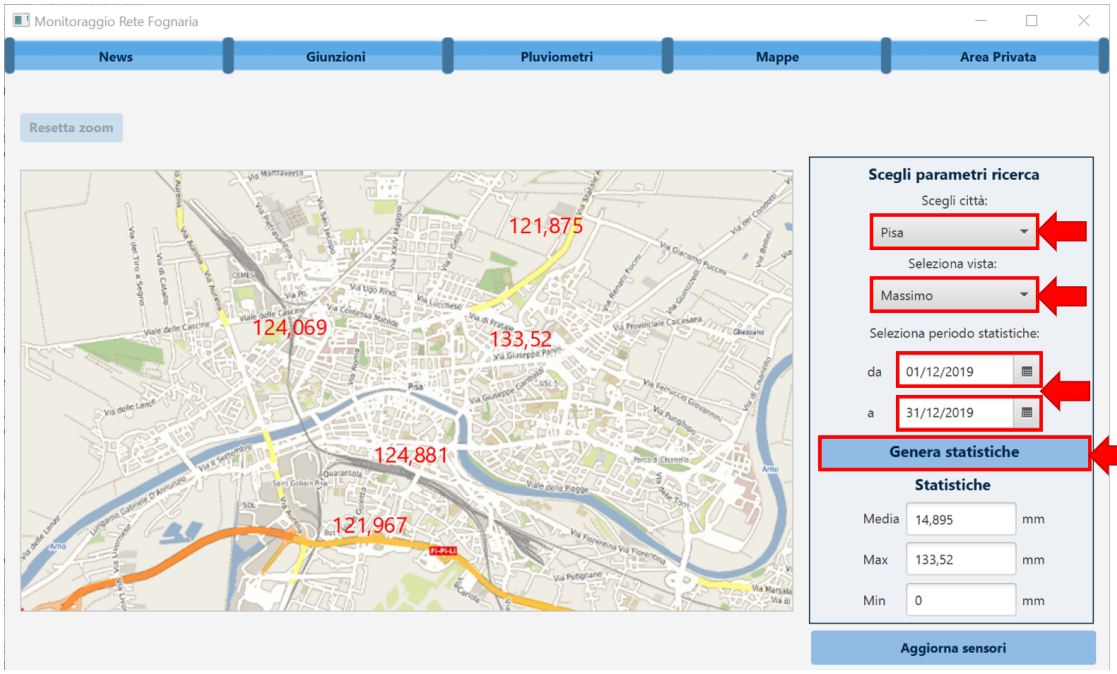


Figura 22: Dati statistici e visualizzazione su mappa

6.7 Area Privata

All'interno dell'Area Privata, l'amministratore può:

- effettuare il logout, premendo il tasto **Logout**
- visualizzare le proprie informazioni e salvare le eventuali modifiche premendo il tasto **Salva**

- aggiornare i valori delle soglie di allerta premendo il tasto **Aggiorna**
- caricare un file *.txt* (contenente i dati simulati relativi ad una giunzione) o *.json* per l'import di qualsiasi tipo di dati su MongoDB. La scelta del file viene effettuata premendo il tasto **Sfoglia** e il caricamento premendo il tasto **Inserisci**. Il file *.txt* deve essere prodotto secondo la formattazione del simulatore SWMM5, esportando manualmente i dati relativi ad una singola giunzione (esempi di file *.txt* formattati sono presenti nella cartella *src/main/resources/DatiSimulatoreGiunzioni*). Per il file *.json* il formato deve essere quello specificato dal formato dei documenti sul database, di cui è riportato un esempio:

  ```
  {"idLuogo":"FGN_PI_10","dataRiferimento": {$date:
  ↪ 1234656000000},"timestampAggiornamento": {$date:
  ↪ 1234560000000},"attributoLIMS": "attributo","campioni":
  ↪ [1.0,2.3,3,4],"qualita": ["B","I","B"],"media":
  ↪ 0.18,"messaggioErrore": "messaggio","simulato": true}
  ```

  Un oggetto JSON non deve avere caratteri di invio, nè spazi tra un attributo e l'altro e per inserire più oggetti contemporaneamente è necessario separare i vari oggetti da un carattere di invio.
- aggiornare i sensori presenti sul database MongoDB dal database *adapt* premendo il tasto **Aggiorna** corrispondente

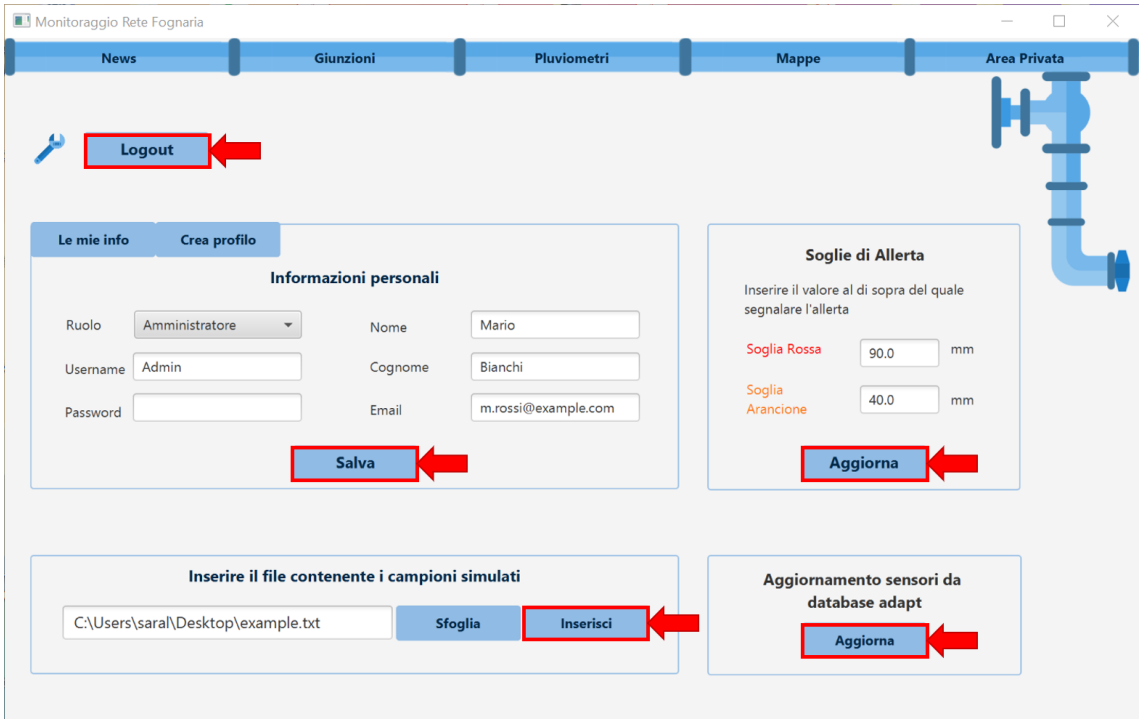


Figura 23: Area Privata Amministratore

L'amministratore, inoltre, può creare un nuovo utente premendo sul pulsante **Crea profilo**, inserendo le informazioni all'interno del form e premendo il tasto **Inserisci**.

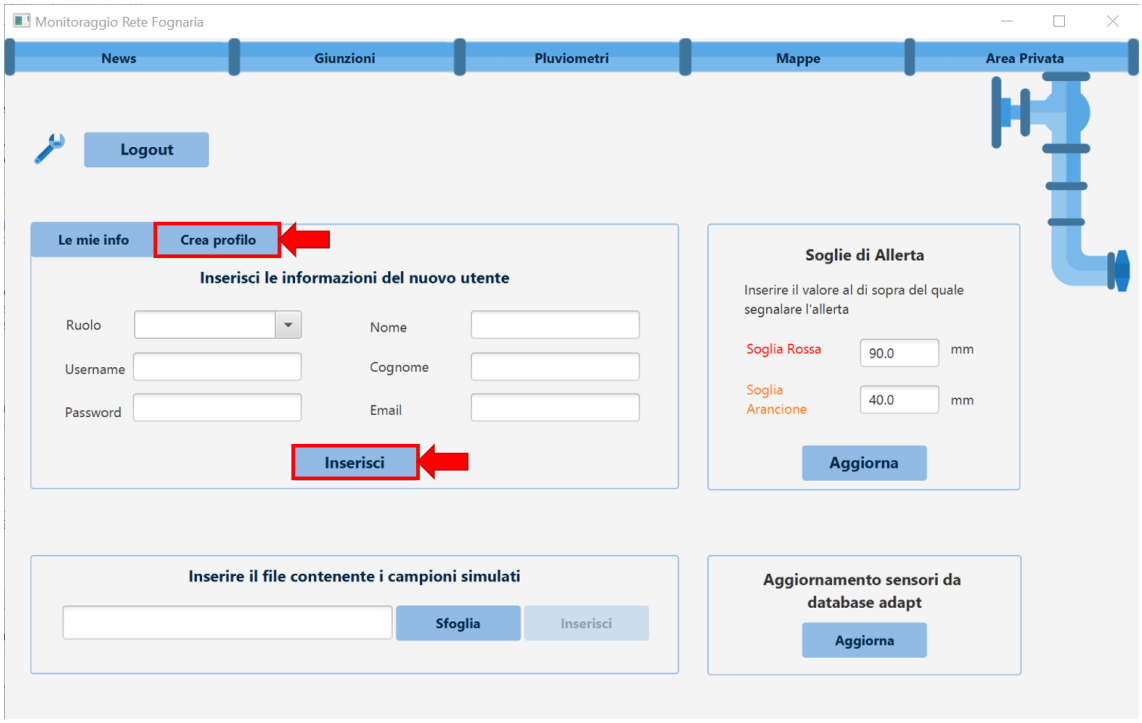


Figura 24: Inserimento nuovo utente

7 Progettazione del Backend

Lato backend l'applicazione deve:

- elaborare ogni nuova previsione riguardo alle precipitazioni nelle successive 48 ore
- individuare, nell'intorno delle due città prese in considerazione, la precipitazione massima attesa
- estrarre la serie temporale relativa alla cella ($2km \times 2km$) in cui e' prevista la maggiore precipitazione
- utilizzarla come intresso al simulatore SWMM (tale previsione sarà pertanto una "wort-case forecast")
- prelevare la serie temporale prodotta dal simulatore per i pozzetti più critici ed inviarle nel database MongoDB

7.1 Caratteristiche dei dati di previsione della pioggia fornite da LAMMA

Il Lamma invia ogni 12 ore la previsione relative alle precipitazioni in formato grib2. La lettura di tali file può essere fatta in Matlab mediante libreria NETCDF.

7.1.1 Descrizioni delle nuove previsioni, ogni 12 ore (2020-)

Una descrizione abbastanza dettagliata delle previsioni che ci vengono fornite da LAMMA mediate invio su server SFTP dell'Università dei Pisa si possono trovare al seguente link: La parte di nostro interesse e' solo quella relativa al modello **Moloch**.

Moloch è un modello numerico di mesoscala creato e sviluppato dal gruppo di modellistica numerica (MODAT) del CNR-ISAC di Bologna per diversi scopi: dalla ricerca in campo della fisica della atmosfera, al supporto per campagne di misura atmosferica internazionali, fino alle previsioni operative a fini di protezione civile.

Il MODAT mantiene ed aggiorna costantemente il modello Moloch, sia in termini di correzione del codice, sia implementandone nuove funzionalità e curando l'emissione di periodici aggiornamenti.

Data	Ora	Dim. (byte)	Nome del file
2020-01-07	17:56	6,163,755	moloch_2015_apcp_daysum_TOS.grib2
2020-01-07	17:56	7,540,388	moloch_2016_apcp_daysum_TOS.grib2
2020-01-07	17:56	8,399,411	moloch_2017_apcp_daysum_TOS.grib2
2020-01-07	17:56	8,232,571	moloch_2018_apcp_daysum_TOS.grib2

In particolare Moloch è un modello non idrostatico e tratta la convezione in maniera esplicita (ossia senza la parametrizzazione dei cumuli), ed è adatto quindi a simulazioni ad alta e altissima risoluzione orizzontale (anche inferiore al km). Ad inizio del 2018 è stata aggiunta la possibilità di effettuare il nesting diretto nei dati dei modelli globali e in se stesso.

Da Marzo 2017 il Consorzio LaMMA, grazie ad una collaborazione tecnico-scientifica con l'ISAC, fa girare operativamente e per scopi di ricerca il modello Moloch sulle proprie macchine di calcolo, con differenti configurazioni e dati iniziali e al contorno.

7.1.2 Tipologia di dati forniti da LAMMA: forecasts e hindcasts

I dati dal 2020 in avanti sono delle emphforecasts (dati iniziali analisi, al contorno forecast), mentre i dati degli anni precedenti (2015-2018, si veda sotto) sono delle *hindcasts*.

Nel caso di hindcasts, i dati iniziali e al contorno sono delle *re-analysis* globali, nelle quali non viene fatta una regional analysis per l'ora di inizializzazione. Invece i dati di *forecasts* (dal 2020 in avanti) sono delle forecast vere e proprie, ossia: i dati iniziali sono delle analisi globali (non re-analysis, ovviamente) e le condizioni al contorno sono anch'esse delle forecasts.

Si noti che c'è da attendersi che le prestazioni delle hindcasts (anni 2015-2018) siano superiori di quelle delle forecasts (anni 2020 e seguenti).

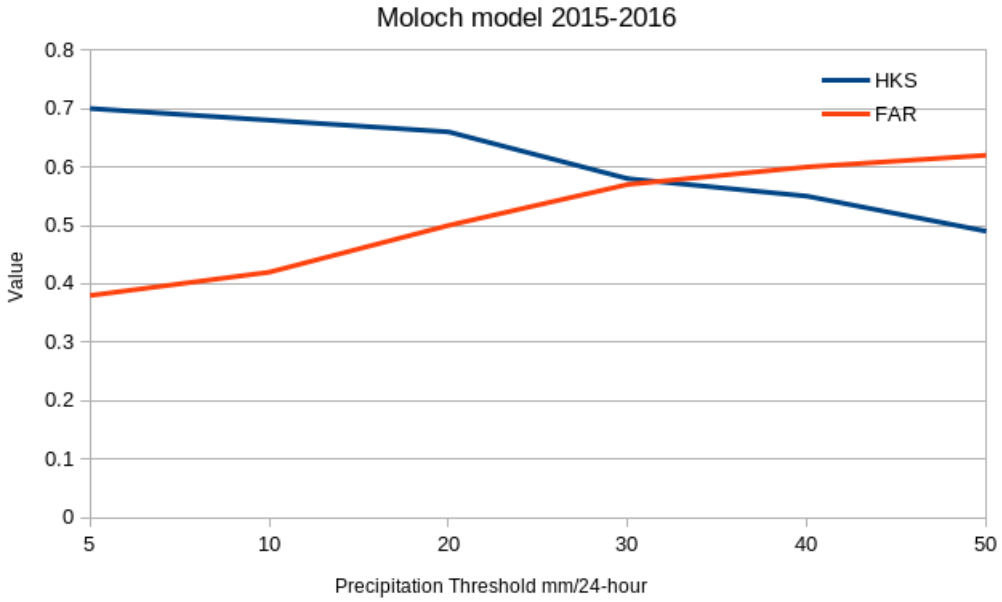
7.1.3 Previsioni anni passati (2015-2019)

Il Lamma ha anche fornito le previsioni, in re-analysis, per il 2015, 2016, 2017 e 2018.

Le previsioni degli anni passati sono interessanti per iniziare a valutare l'accuratezza che potremo attenderci, nella generazione di allarmi di potenziale allagamento, per i prossimi anni.

In particolare le simulazioni per gli anni 2015-2018 sono state generate alle ore 18.00 del giorno precedente ciascun giorno di previsione. La singola previsione ha durata 30 ore, con boundary conditions aggiornate ogni 6 ore. Maggiori dettagli sono disponibili al seguente link: `https://www.ecmwf.int/sites/default/files/special_projects/2018/spitbran-2018-report1.pdf`

In figura sono riportati due skill scores per il modello Moloch, in modalità hindcast (valori aggregati per gli anni 2015 e 2016).



Le previsioni per il 2019 non sono ancora disponibili. Il motivo è interessante ed è il seguente. Il LAMMA ha bisogno delle unità di calcolo (SBUs) del supercomputer ECMWF. Al momento stanno attendendo l'approvazione delle ulteriori SBUs per il nostro ADAPT. Pertanto tali previsioni saranno disponibili a metà 2020. L'altro problema è che attualmente sul MARS catalogue non sono presenti i dati di inizializzazione ERA5 per i mesi di Novembre e Dicembre 2019 e pertanto Lamma deve aspettare che vengano rilasciati prima di procedere con il run dei modelli ad alta risoluzione.

8 Conclusioni e Sviluppi Futuri

Nel presente documento è stata progettata l'intera applicazione per il monitoraggio della rete di raccolta delle acque della rete fognaria unitaria in ambito urbano.

L'applicazione è stata suddivisa in tre sottosistemi: sistema database, sotto-sistema Front-End (ossia l'interfaccia grafica), e sotto-sistema Back-End (ossia motore di elaborazione delle previsioni LAMMA e di calcolo dei livelli attesi nelle varie giunzioni critiche, mediante uso di modello SWMM).

Il sistema progettato nel seguente documento potrà essere esteso aggiungendo le seguenti funzionalità:

- Implementare nell'interfaccia la possibilità di vedere i sensori che sono stati installati nel passato, ma sono stati sostituiti.
- Prevedere un import diretto dal database relazionale al database non relazionale, spostando il carico computazionale dall'applicazione verso il server.
- Prevedere un'integrazione diretta tra i dati generati dai sensori e il caricamento sul database non relazionale.
- Implementare lo *sharding geografico* (ossia una replica geografica del server, che aumenta prestazioni e affidabilità), qualora fossero presenti server collocati nelle vicinanze delle zone poste sotto osservazione.
- Aggiunta del calcolo del tempo di ritorno e sua visualizzazione nell'interfaccia grafica
- analisi dell'attendibilità delle previsioni LAMMA, alla luce delle hindcast degli anni 2015-2019.