"Software Engineering"

Course

a.a. 2018-2019

Template version 1.0

Deliverable #3

Lecturer: Prof. Henry Muccini (henry.muccini@univaq.it)

Dashboard Monitoraggio Ambientale

Date	20/01/2019
Deliverable	Documento Finale – D3
Team (Name)	Soft_Ing

Team Members		
Name & Surname	**Matriculation Number**	**E-mail address**
Salvatore Salernitano	*242016*	*salvatore.salernitano@student.univaq.it*
Lorenzo Salvi	*242387*	*lorenzo.salvi@student.univaq.it*
Ludovico Di Federico	*243542*	*ludovico.difederico@student.univaq.it*
Francesco Catani	*246186*	*francesco.catani@student.univaq.it*
Marco Poscente	*243591*	*marco.poscente@student.univaq.it*

Table of Contents of this deliverable

Sommario

List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Ristrutturazione Modello ER	15/12/18	10/01/2019	Il team ha deciso di inserire come Foreign Key nella relazione Sensore, il campo Edificio ed Area, in modo tale da avere maggior tracciabilità dei sensori all'interno di un piano, oppure all'interno di un Edificio, o all'interno di un'Area.
Testing Prototipo e correzione di eventuali bug	10/01/2019 16/01/2019	10/01/2019 16/01/2019	Il Team ha deciso di far testare il prototipo al cliente per migliorare l'usabilità del software.
Implementazione FR Aggiunta Sensore e SetBackup	10/01/2019	12/01/2019	Conclusa l'implementazione della funzione Aggiunta Sensore e setBackup per la Dashboard Gestore. Nello specifico, sono state modificate le seguenti componenti per permettere la creazione di un nuovo sensore e, nello stesso momento, l'aggiunta del suo rispettivo Backup: *View*: CreazioneSensoreBackup.java; *Controller*: GestoreSensoreController.setSensore; *Model*: Sensore.setBackup, GestoreDati.setSensore.
Implementazione FR Ripristino Sensore	15/01/2019	15/01/2019	Conclusa l'implementazione della funzione ripristino() per ripristinare i valori ambientali di un sensore mediante l'utilizzo della Dashboard Gestore Sensori. *View*: RipristinoSensore.java; *Controller*: GestoreSensoreController.ripristino(ID); *Model*: GestoreDati.ripristino(ID).
Implementazione FR Invio Segnale	15/01/2019	15/01/2019	Conclusa l'implementazione della funzione setSegnale() per effettuare un invio di un segnale da parte di un sensore, situata nella Classe Sensore.

A. Requirements Collection

A.0 Detailed Scenarios

SCENARIO 1: PRIMO AVVIO: Mediante la Interfaccia Login, il *Gestore* dei sensori o l'Amministratore potrà accedere alla sua area riservata utilizzando le sue credenziali d'accesso costituite da Username e Password. Nel caso dell'Amministratore, una volta loggato alla Dashboard Admin, l'utente visualizzerà quattro bottoni: Ticket, *Recupero, Aggiungi e Logout*. Nel caso del Gestore dei Sensori, dopo aver eseguito l'accesso alla propria Dashboard, visualizzerà i seguenti bottoni: *Ticket, Monitoraggio, Aggiungi Sensore e Logout*.

SCENARIO 2: INVIO SEGNALE: *Il sensore invia un segnale contenente le sue variabili ambientali e il suo stato di funzionamento.* Il sensore sarà evidenziato a seconda dei valori ambientali che risultano fuori soglia, nello specifico, se un sensore risulta avere uno o più valori fuori soglia, esso comparirà nella sezione "Sensori a Rischio (Da Ripristinare)" nella dashboard gestore di sua appartenenza. Se il sensore risulterà in questa sezione, invierà più frequentemente il segnale. Per i valori ambientali dei sensori meno rilevanti, il Gestore potrà visualizzare le informazioni mediante il bottone MONITORAGGIO (dove, comunque, sarà possibile visualizzare la lista di tutti i sensori appartenenti al quel Gestore specifico).

SCENARIO 3: RIPRISTINO: Il Gestore dei Sensori effettuerà un backup dei valori ambientali nel momento in cui aggiungerà un nuovo sensore. Tali valori che verranno salvati all'interno del Database, potranno essere utilizzati dal Gestore dei sensori per poter ripristinare i valori ambientali in caso di dati fuori soglia, mediante il bottone RIPRISTINA UN SENSORE, presente nella Dashboard Gestore Sensori. L'utente dovrà passare l'ID del sensore che vuole ripristinare e per visualizzare l'effettivo cambiamento, dovrà uscire dalla propria schermata ed effettuare nuovamente il Login. In parole povere, una volta che un Sensore viene ripristinato non comparirà più nella lista dei Sensori a rischio.

SCENARIO 4: TICKET: ***Sia l'utente Amministratore sia l'utente Gestore dei sensori, presentano un bottone denominato TICKET, che rispettivamente svolgerà ruoli differenti.*** Il Gestore dei Sensori potrà inviare un Ticket all'Amministratore mediante il bottone TICKET, situato nella sua Dashboard, nel caso in cui esso abbia bisogno di un feedback per risolvere un determinato problema, ad esempio: l'aggiunta di un nuovo Gestore attinente alla propria area geografica, oppure chiarimenti attinenti all'utilizzo del Software. D'altra parte, l'Amministratore visualizzerà i Ticket all'interno di una specifica lista, mediante il bottone TICKET situato nella relativa Dashboard, e risponderà ad essi mediante il bottone RISPOSTA dove modificherà la descrizione dei feedback inviati dai Gestori. Infine, il Gestore dei Sensori, potrà constatare la modifica della descrizione fatta dall'Admin, all'interno della specifica sezione TICKET.

A.1 Functional Requirements

Per modellare la complessità del nostro Sistema, il team ha optato per la **decomposizione funzionale** che consiste nel suddividere il Sistema in base alle sue funzionalità:

- **Dashboard**: interfaccia grafica utilizzata dai gestori dei sensori e dall'Amministratore del Sistema per avere una panoramica dettagliata dell'intero funzionamento del Sistema. Sono presenti due tipologie di Dashboard: ***Dashboard Admin*** dove l'Amministratore potrà interagire con i Gestori (tramite Ticket) e si occuperà del corretto funzionamento, ad esempio, permette l'aggiunta di nuove figure tecniche come Gestori o gli stessi Admin; ***Dashboard Gestore*** dove il Gestore potrà controllare l'operato dei sensori (nel caso in cui i loro valori risultino fuori soglia o meno) attinenti alle loro aree geografiche e, in più si occuperà di ripristinare e salvare nel database i valori del sensore selezionato;
- **Invio Segnali**: il nostro Sistema deve permettere ai sensori di inviare periodicamente tutte le loro informazioni ambientali compreso lo stato di funzionamento. Nello specifico queste informazioni saranno visualizzate nella Dashboard Gestore dei Sensori attinente al Gestore di sua competenza. ***Nel dettaglio vedere lo Scenario 2;***
- **Ripristino parametri sensore**: il Sistema deve garantire al Gestore dei Sensori di ripristinare i parametri dei sensori in caso in cui si riscontrano dei valori fuori soglia o i sensori risulti essere malfunzionanti. Nello specifico, cliccando sul sensore che si vuole ripristinare, si potrà effettuare l'operazione sopra indicata. ***Nel dettaglio vedere lo Scenario 3***;
- **Aggiunta del Sensore e backup valori ambientali**: Il Gestore dei Sensori deve essere in grado di effettuare il backup dei parametri del sensore nel momento in cui si aggiunge e si configura lo stesso;
- **Interazione**: l'Amministratore del Sistema e i Gestori dei Sensori devono interagire tra loro (es: mediante utilizzo dei ticket). In dettaglio, l'interazione prevede: da una parte il Gestore dei Sensori crea ed invia un ticket all'Amministratore, dall'altra parte l'Amministratore visualizzerà il ticket che gli è stato inviato dal Gestore e risponderà modificando la descrizione del ticket stesso;
- **Registrazione Gestore Sensori e Amministratore**: L'Amministratore del Sistema può inserire un nuovo Gestore dei Sensori ed un nuovo Amministratore assegnandogli delle credenziali d'accesso (Username e Password);

A1.1 Use Case Diagrams

Il team ha deciso di utilizzare **Draw.io** per modellare lo Use Case Diagram:

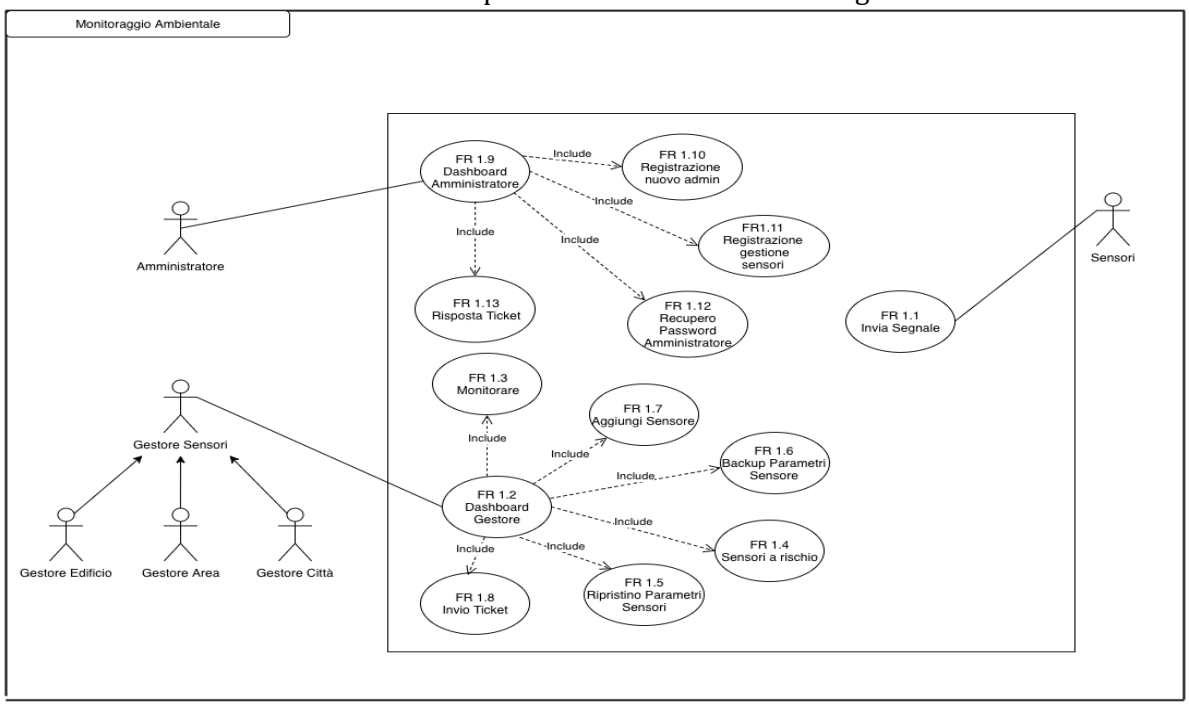


Fig.0: Use-Case Diagram Monitoraggio Ambientale

ATTORI

1. **Sensori,** dispositivo elettronico con lo scopo di collezionare variabili ambientali (ad esempio, temperatura, luminosità, pressione, umidità) appartenenti ad una determinata area geografica. Il Sensore sarà posizionato in un piano all'interno di un edificio, nelle vicinanze dell'edificio e nelle aree esterne adiacenti (ad esempio piazze, strade, incroci, spazi comunali);
2. **Gestore Sensori**, persona che si occupa della gestione (Dashboard Monitoraggio Ambientale) e della corretta funzionalità dei sensori inerenti ad una determinata area geografica.
   I Gestori dei Sensori saranno generalizzati a seconda dell'area geografica dove operano (ad esempio, **gestore di un edificio, gestore di un'area, gestore dell'intera città**);
3. **Amministratore**, colui che attraverso la propria interfaccia, si occuperà di eseguire operazioni quali: registrare un Gestore dei Sensori o un Amministratore, recuperare credenziali (Username, Password) di un Admin ed interagire tramite Ticket con i Gestori dei Sensori.

USE-CASES

- **FR 1.1**: Il sensore invia periodicamente i segnali (informazioni ambientali e stato di funzionamento);
- **FR 1.2**: Il Gestore Sensori attraverso la Dashboard Gestore potrà effettuare operazioni di controllo, ripristino ed aggiunta di sensori all'interno di un'area specifica ed interagire con l'Amministratore mediante uso di Ticket;
- **FR 1.3**: Il Gestore Sensori, può selezionare un'area geografica e *visionare i sensori* inerenti all'area d'interesse con tutte le loro informazioni oppure visionare un

determinato Sensore. Nello specifico un Gestore Edificio potrà visionare i sensori di un piano, un Gestore Area potrà visionare i sensori di un edificio e un Gestore Città potrà visionare i sensori di un'area;

- **FR 1.4**: Vengono segnalati i sensori che *presentano valori fuori soglia*. Nello specifico i Sensori con più valori fuori soglia saranno segnalati in una lista, visibile dal Gestore appena effettua il login alla propria area riservata;
- **FR 1.5**: Il Gestore Sensori potrà *ripristinare* i valori fuori soglia di un sensore che ha selezionato, sostituendoli con i valori ambientali di Backup;
- **FR 1.6**: Il Gestore Sensori potrà *effettuare il Backup* dei parametri ambientali di un Sensore, nel momento in cui verrà aggiunto un nuovo Sensore inerente alla sua area di competenza;
- **FR 1.7**: Il Gestore Sensori potrà *aggiungere un nuovo Sensore* impostando i valori attuali e quelli di Backup;
- **FR 1.8**: Il Gestore Sensori potrà *inviare un Ticket* ad un Amministratore contenente un feedback con lo scopo di risolvere un determinato problema;
- **FR 1.9**: L'Amministratore attraverso la Dashboard Amministratore potrà effettuare operazioni di *recupero delle credenziali* di un Admin, *aggiunta* di un Gestore o Admin e *interagire* con un Gestore, modificando il Ticket ricevuto;
- **FR 1.10**: L'amministratore del sistema *potrà aggiungere* un nuovo amministratore assegnando delle nuove credenziali (es: username e password);
- **FR 1.11**: L'amministratore del sistema potrà aggiungere un nuovo Gestore Sensori assegnando delle nuove credenziali (es: username e password) e assegnandogli una Tipologia (Gestore Edificio, Gestore Area, Gestore Città);
- **FR 1.12:** L'Amministratore del sistema può recuperare l'username e password di un altro Amministratore mediante la chiave di recupero, associata alle credenziali da recuperare appartenenti all'appropriato amministratore;
- **FR.1.13** L'Amministratore potrà rispondere ad un Ticket, inviatogli da un Gestore, visualizzando la lista appropriata e quindi, modificando la descrizione del ticket selezionato;

A1.2 Tabular description of the most relevant use case

Il Team ha deciso di dare priorità ai seguenti Use-Case:

FR 1.2

USE CASE	Dashboard Gestore	
Goal in Context	Il Gestore Sensori può effettuare operazioni di controllo, ripristino ed aggiunta di sensori; interagire con l'admin mediante ticket.	
Scope & Level	Questo Use Case ha lo scopo di permettere al Gestore Sensori di svolgere le operazioni che sono stati precedentemente	

	specificati nella “Goal in Context”	
Preconditions	Ci aspettiamo che il Gestore Sensori in maniera del tutto sicura e rapida possa entrare nella dashboard. Ci aspettiamo che il Gestore Sensori abbia le credenziali di accesso alla sua area riservata.	
Success End Condition	Grazie a questo Use Case il gestore sensori può svolgere molte operazioni all’interno della dashboard, quali monitoraggio dei sensori all’interno della sua area di competenza ed altre funzioni come il ripristino dei valori ambientali di un sensore, aggiungere ed effettuare il backup degli stessi, inviare un ticket e visualizzare i sensori correttamente funzionanti.	
Failed End Condition	Un mancato funzionamento della dashboard andrebbe a compromettere il corretto funzionamento dei Sensori di sua responsabilità.	
Primary, Secondary Actors	Gestore Sensori Gestore Edificio, Gestore Area, Gestore Città.	
Trigger	Il gestore di un determinato sensore effettua il login nella propria area riservata attraverso la Dashboard.	
DESCRIPTION	Step	Action
	1	Inserire Username
	2	Inserire Password
	3	Accesso alla Dashboard
EXTENSIONS	Step	Branching Action
	1a	-Monitoraggio dei Sensori (Lista Sensori di sua competenza ed ricerca di un Sensore mediante id anche se non è di sua competenza).

		-Visualizzare i sensori a rischio (Sensori da ripristinare) -Aggiunta e backup del sensore -Invio ticket -Ripristino Parametri Sensori.
SUB-VARIATIONS		Branching Action
	1	In base alla tipologia del gestore che opera nella Dashboard verranno evidenziati i parametri dei sensori più adeguati.

FR 1.9

USE CASE	Dashboard Amministratore	
Goal in Context	L'amministratore potrà effettuare determinate operazioni dalla propria dashboard quali registrazione nuovo admin e gestore sensori, recupero credenziali admin, rispondere ai ticket inviati dai gestori.	
Scope & Level	L'amministratore effettua un'operazione di assistenza ai gestori come registrarne di nuovi ed inviare dei feedback mediante ticket.	
Preconditions	Ci aspettiamo che l'amministratore in maniera del tutto sicura e rapida possa entrare nella dashboard. Ci aspettiamo che l'admin abbia le credenziali di accesso alla sua area riservata e che conosca la chiave di recupero nel caso in cui esso non possa accedere alla dashboard.	
Success End Condition	Grazie a questo use case l'amministratore può svolgere molte operazioni all'interno della dashboard quali aggiunta di un admin o gestore, recupero	

	credenziali admin, rispondere ad eventuali ticket.	
Failed End Condition	Una mancato funzionamento della dashboard e delle sue funzionalità potrebbe, nel lungo periodo, rendere il sistema inaccessibile agli altri utenti.	
Primary, Secondary Actors	Amministratore del sistema, Gestore dei Sensori.	
Trigger	L'amministratore deffettua il login nella propria area riservata attraverso la Dashboard.	
DESCRIPTION	Step	Action
	1	Inserire Username
	2	Inserire Password
	3	Accesso alla Dashboard
EXTENSIONS	Step	Branching Action
	1a	-Registrare un nuovo admin; -Registrare un nuovo gestore; -Recuperare credenziali di un admin -Rispondere ad eventuali ticket
SUB-VARIATIONS		Branching Action
	1	<list of variation s>

A1.1 GUI Requirements (da riempire a partire dalla Versione 2)

1. La GUI deve permettere all'utente (Gestore dei Sensori o Amministratore) di accedere al Sistema Software inserendo la propria username e password:

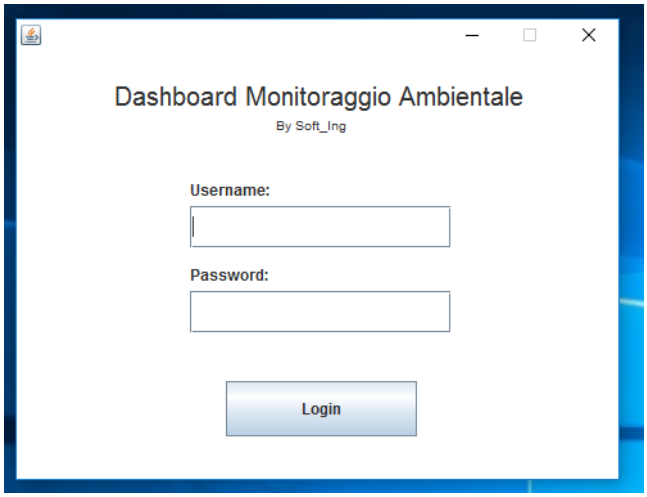


Fig. 1: Interfaccia Grafica della Login.java

2. Nel caso in cui non vengano inseriti l'username e la password verrà visualizzata una finestra dove si chiederà di inserire obbligatoriamente i dati per accedere alla Dashboard richiesta:

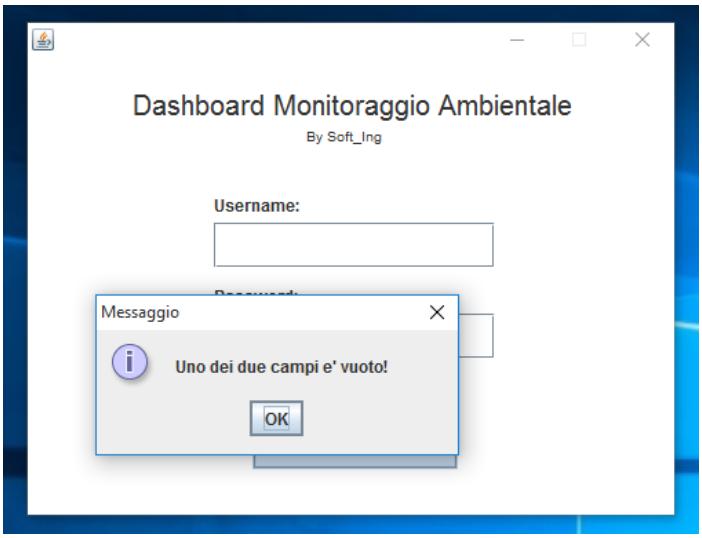


Fig. 2: Interfaccia Grafica Messaggio della Login.java

3. La Dashboard Admin deve permettere all'Amministratore di aggiungere un nuovo Admin e/o Gestore dei Sensori, recuperare l'username e la password di un altro Amministratore mediante il bottone "Recupero", leggere e rispondere ai ticket inviati dai Gestori dei Sensori. Infine il pulsante "Logout" permette all'utente di chiudere la sessione corrente e quindi di ritornare alla schermata Login, "Dashboard Monitoraggio Ambientale":

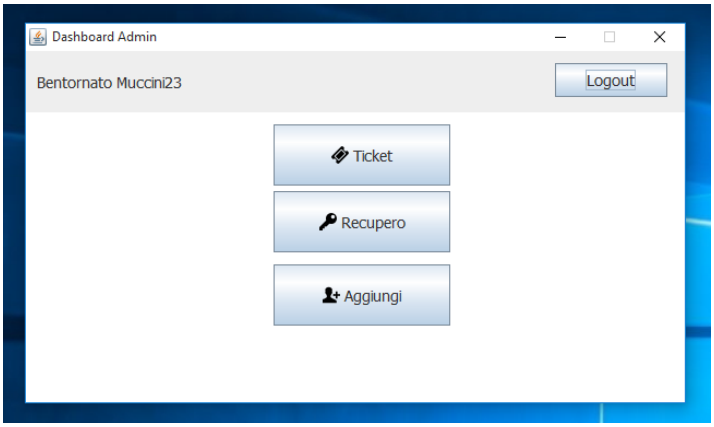


Fig. 3: DashboardAdmin.java

4. Cliccando sul bottone “Ticket” verrà mostrata la finestra contenente la lista dei Ticket ricevuti dell’amministratore:

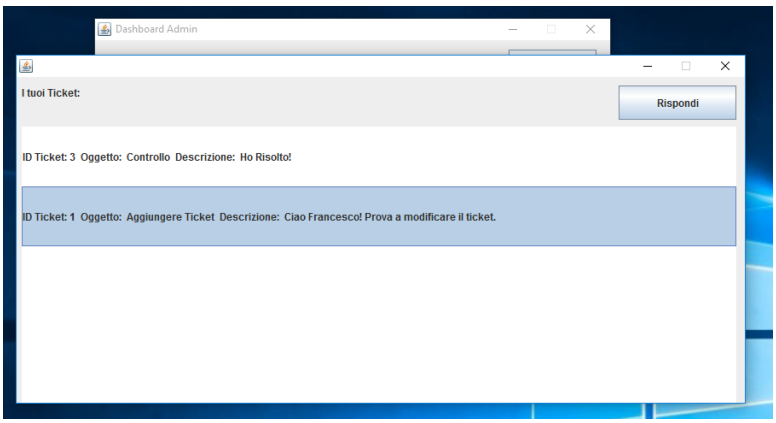


Fig. 4: Interfaccia Lista AdminTicket.java

5. Selezionando un ticket dall’elenco e cliccando su “rispondi” sarà mostrata una finestra di dialogo dove sarà possibile inserire una risposta:

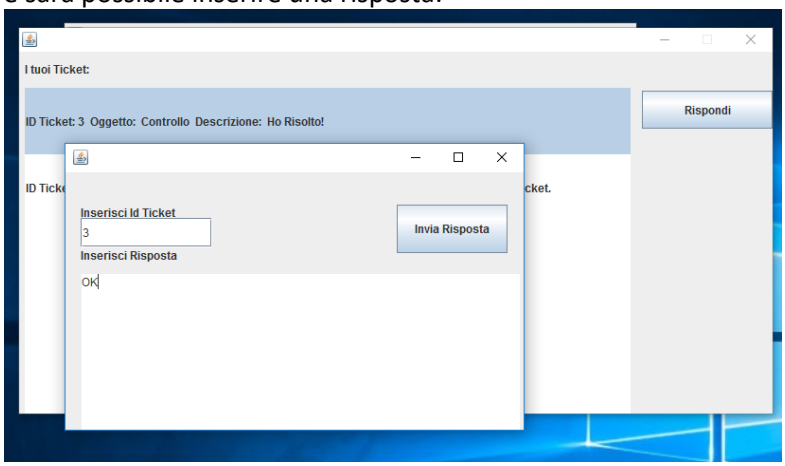


Fig. 5: Finestra RispondiTicket.java

6. Cliccando il bottone “Recupero” sarà possibile visualizzare una finestra per il recupero delle credenziali di un Admin tramite chiave di recupero:

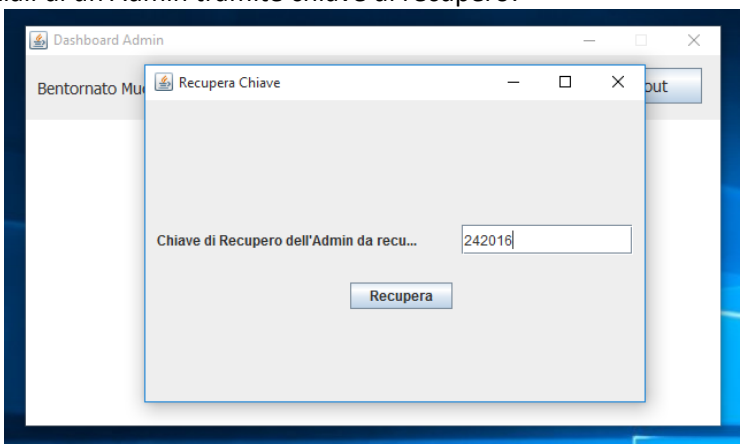


Fig. 6: RecuperaChiave.java

7. Dopo aver inserito la chiave di recupero ed aver cliccato “recupera” verrà mostrata un messaggio contenente le credenziali richieste:

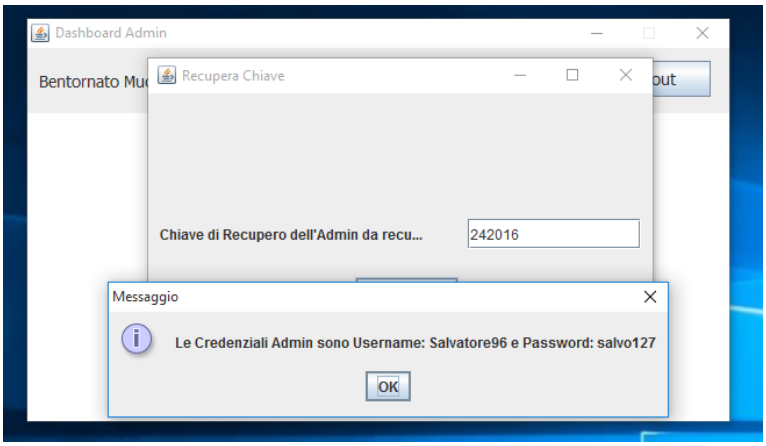


Fig. 7: Messaggio credenziali recuperate

8. Cliccando sul bottone “Aggiungi”, verrà mostrata una finestra dove sarà possibile aggiungere un nuovo Admin o Gestore dei Sensori:

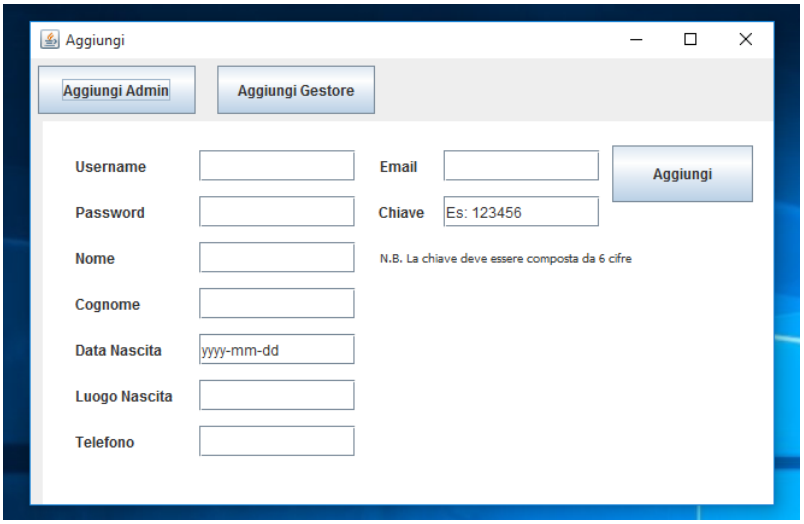


Fig. 8: Interfaccia Grafica Aggiungi.java (Admim)

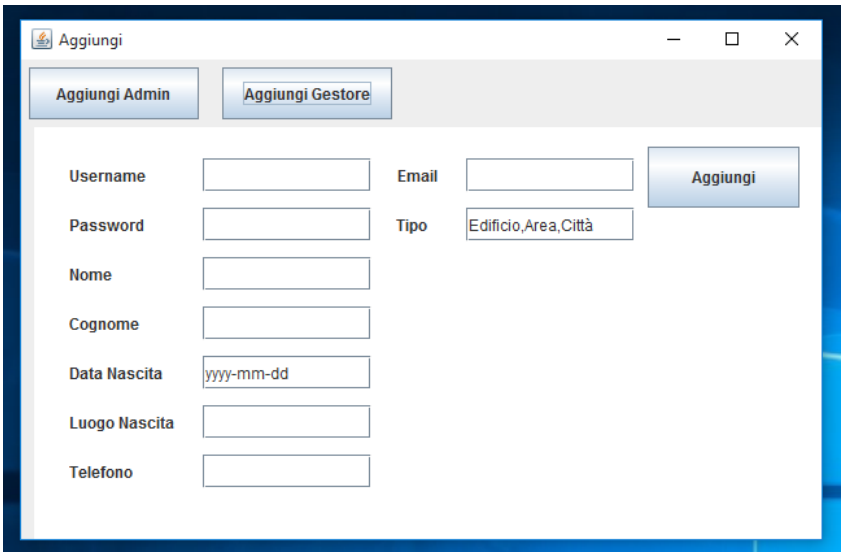


Fig. 9: Interfaccia grafica Aggiungi.java (Gestore)

9. La Dashboard Gestore Sensori deve permettere al Gestore di monitorare i sensori di sua competenza e visualizzare le informazioni di un sensore che sta cercando, inviare i ticket agli amministratori nel caso in cui egli ha dei problemi di varia natura, aggiungere un nuovo sensore ed effettuare un backup, visualizzare i sensori a rischio (da ripristinare) e, infine, ripristinare il sensore. Infine il pulsante "Logout" permette all'utente di chiudere la sessione corrente e quindi di ritornare alla schermata Login, "Dashboard Monitoraggio Ambientale":

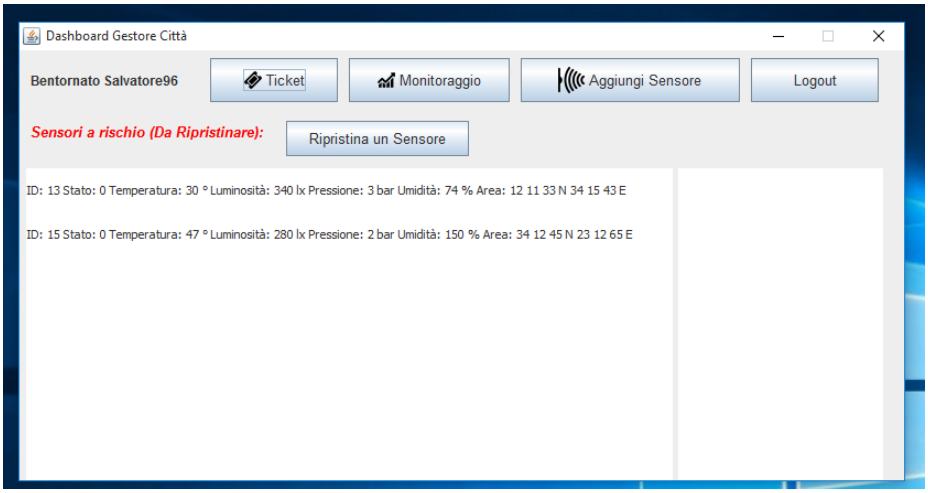


Fig. 10: DashboardGestore.java

10. Cliccando il tasto "Monitoraggio", si aprirà una sottofinestra laterale dove sarà possibile selezionare un sensore (anche non di sua competenza) ed accedere alle sue informazioni verificando anche il Gestore che gestisce quel determinato sensore, oppure visualizzare la lista dei sensori di sua competenza:

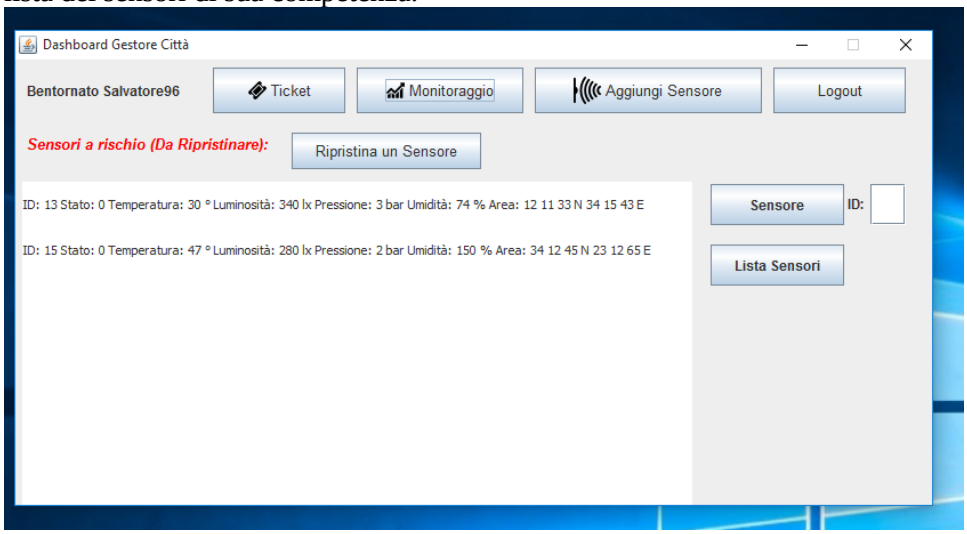


Fig. 11: Dashboard Gestore dei Sensori Bottone Monitoraggio

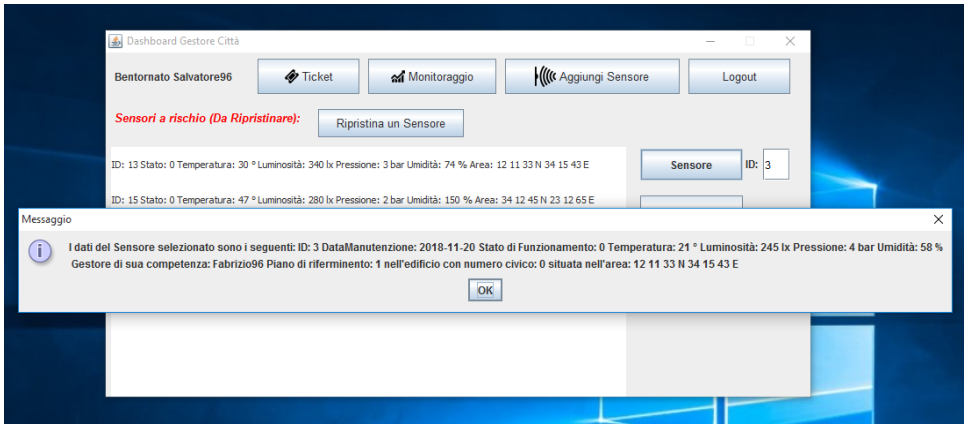


Fig. 12: Visualizzazione delle informazioni di un Sensore mediante il bottone Sensore

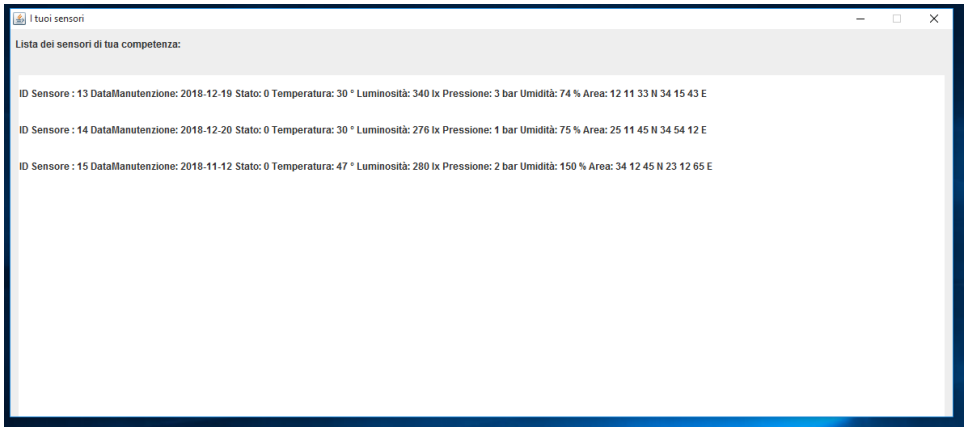


Fig.13: Visualizzazione della Lista dei Sensori di sua competenza

11. Il Gestore potrà aggiungere un nuovo sensore e, nello stesso momento, effettuare il suo backup mediante il bottone “Aggiungi Sensore e Backup”. Nella sezione Backup sono già riportati dei valori consigliati per indicare i range di valori che il Sensore non deve superare. E’ molto importante avere i valori di Range perchè permettono di poter distinguere i sensori con valori fuori soglia “Sensori a Rischio” dai Sensori con valori a norma. I valori ottimali permettono ai Gestori di riportare i Sensori a norma mediante la

sezione Ripristino:

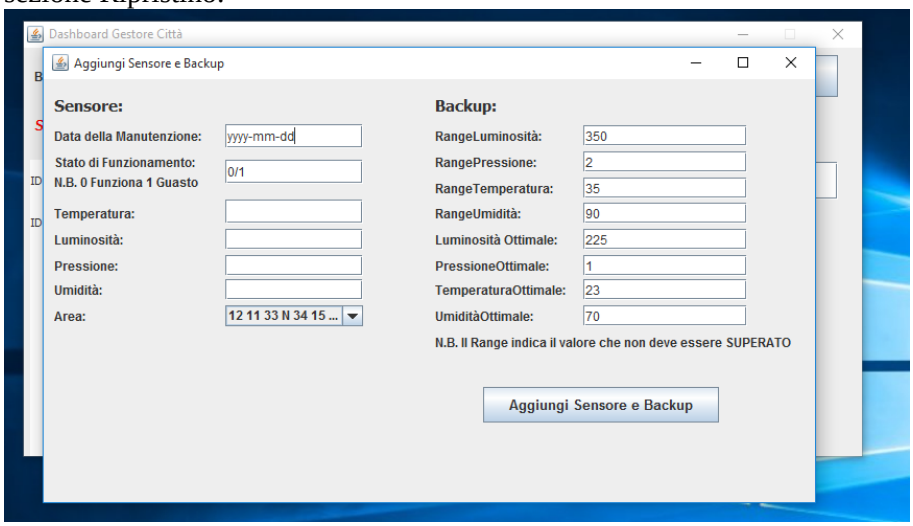


Fig.14: Aggiunta di un Sensore e del suo Corrispondente Backup

12. Mediante la sezione Ticket il Gestore dei Sensori potrà visualizzare i Ticket che ha inviato ad un Admin e crearne degli altri. Questi Ticket hanno il compito di informare l'Admin, nel caso in cui il Gestore abbia bisogno di chiarimenti:

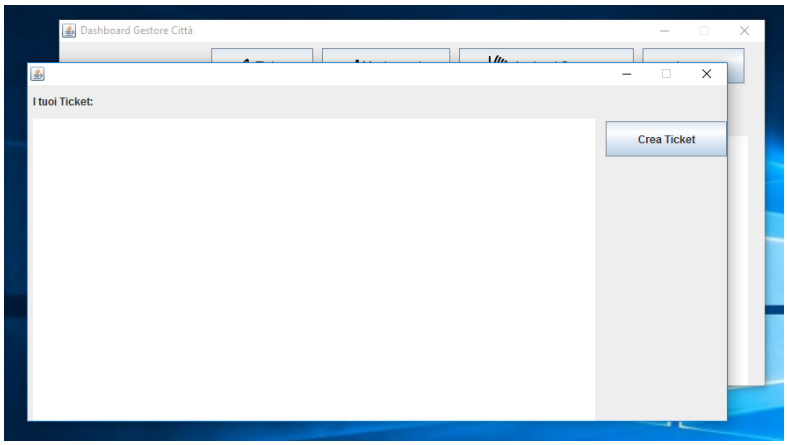


Fig. 15: Lista dei Ticket inviati ad un Admin

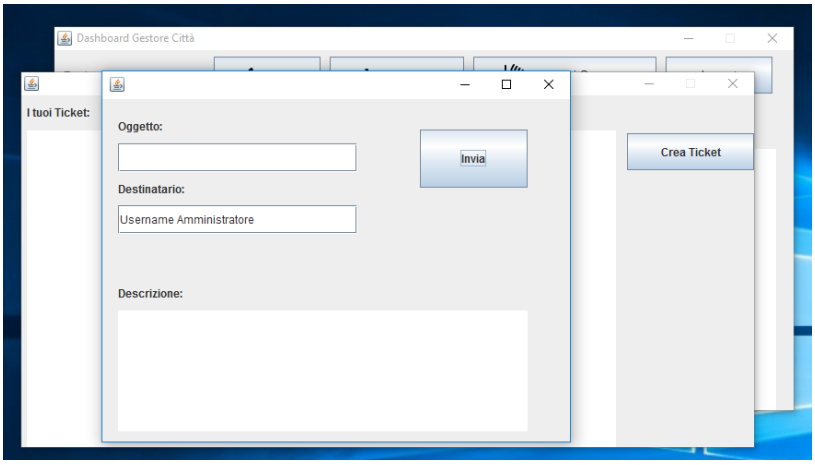


Fig. 16: Creazione di un Ticket da inviare

13. Mediante il bottone "Ripristina un Sensore" il Gestore potrà ripristinare un Sensore passato nella casella di testo. Appena si ripristina un Sensore, verrà visualizzato un messaggio di dialogo che avviserà il Gestore dei Sensori che, appunto, il Sensore è stato ripristinato:

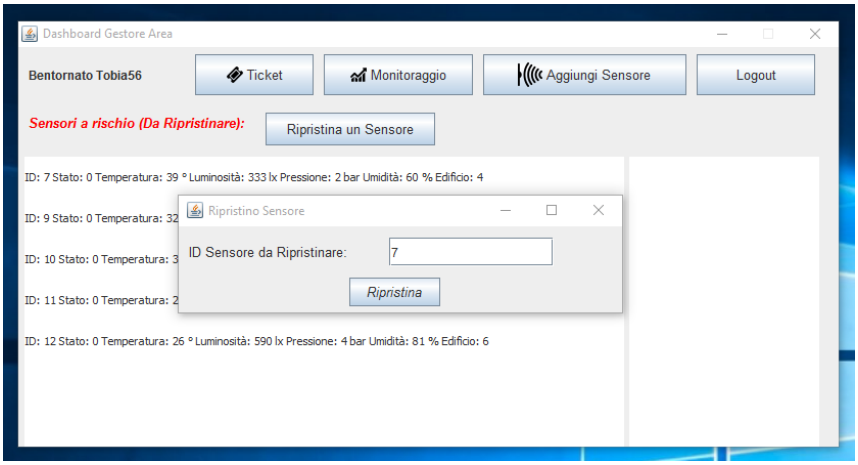


Fig. 17: Procedura di Ripristino del Sensore a rischio

A1.2 Business Logic Requirements (da riempire a partire dalla Versione 2)

1. Deve implementare una funzione per salvare il segnale nel database;
2. Deve implementare una funzione per permettere al Gestore dei Sensori di effettuare il backup dei sensori;
3. Deve implementare una funzione con il quale l'amministratore del sistema può inserire un nuovo gestore dei sensori o amministratore;
4. Decide le regole per l'uso del sistema: tramite il controller prende in input i dati passati dall'utente (es. Mediante l'utilizzo di Textbox e Button) e richiama il metodo appropriato presente nel model che elabora la richiesta dell'utente;
5. Deve implementare una funzione che garantisce al gestore dei sensori di ripristinare i parametri dei sensori nel caso in cui le variabili ambientali dei sensori risultino fuori soglia;

6. Deve essere possibile recuperare le credenziali mediante richiesta fatta all'Admin, passandogli una chiave di recupero;
7. Deve essere possibile gestire, visualizzare e monitorare determinati sensori in una certa area geografica;
8. Deve essere in grado manipolare i feedback espressi dal Gestore dei Sensori tramite l'utilizzo di ticket inviati ad un Admin.
9. Deve implementare una funzione che permette all'utente (Gestore o Admin), attraverso il pulsante Logout, di uscire in modo corretto dalla sessione attuale e quindi di tornare alla schermata della Login.

A1.3 DB Requirements (da riempire a partire dalla Versione 2)

1. Il Database deve essere in grado di memorizzare i dati relativi ai segnali dei sensori, disporre di uno Storage nel quale immagazzinare tali dati senza correre il rischio che vengano persi.
2. Il Database deve essere sempre operativo.
3. Deve essere capace di rispondere e gestire tutte le richieste mandate dai servizi del sistema nel minor tempo possibile.

A.2 Non Functional Requirements

N.F.R. 1 DEPENDABILITY:

- ***N.F.R. 1.1 FAULT TOLERANCE****: Il sistema deve continuare ad offrire i servizi anche se si è persa la funzionalità di una componente. (Esempio: In un piano di un edificio, devono essere presenti almeno due sensori in modo tale da garantire il corretto monitoraggio del piano anche nel caso in cui un sensore risulti danneggiato/fuori uso; sono necessarie almeno 50 postazioni con diritti di amministratore o gestore sensore per garantire almeno un accesso da parte di uno di essi così da poter coprire le proprie mansioni nel caso in cui una postazione risultasse non funzionante).*
- ***N.F.R. 1.2 SECURITY****: Il sistema deve essere in grado di proteggersi da attacchi esterni, accidentali o intenzionali. La security diventa un requisito essenziale, visto che il nostro sistema dovrà essere connesso alla rete per permettere ai sensori, ai gestori dei sensori e all'amministratore di inviare dati rilevanti, come ticket, segnali. (Esempio: l'interfaccia login ed il pulsante logout permetteranno all'amministratore ed al gestore dei sensori di effettuare un accesso ed una disconnessione sicura).*

N.F.R. 2 SCALABILTY*: Il sistema garantisce un'architettura scalabile per supportare future espansioni. (Esempio: L'aggiunta di un sensore, gestore dei sensori, di un admin e di un backup)*

N.F.R. 3 USABILITY*: Il sistema deve offrire una user-friendly-experience. Ovvero, deve risultare facile da utilizzare. (Es: Utilizzo intuitivo delle dashboard)*

N.F.R. 4 PERFORMANCE*: Il sistema deve risultare efficiente e deve lavorare con tempi di esecuzione accettabili, anche nel caso in cui ci siano più accessi in parallelo dei gestori. (Esempio: Invio di 150.000 segnali al minuto e l'accesso in parallelo di almeno 50 gestori).*

A.3 Excluded Requirements

Il team ha deciso di escludere le seguenti funzionalità:

- **Comunicazione tra ditta che utilizza il nostro sistema software con l'Amministratore:** non è di nostra competenza conoscere il modo con cui la ditta chiede di creare *username* e *password* di un nuovo Gestore di Sensori (es. E-mail, fax, recapito telefonico). L'unica cosa di nostra competenza è assegnare *username* e *password* ai gestori per permettergli di accedere alla dashboard.
- **Fase realizzativa dei sensori in maniera fisica:** non è di nostra competenza la realizzazione e manutenzione del sensore o parti di esso. L'unica competenza è la configurazione delle variabili ambientali del sensore mediante il backup del sensore nel momento in cui viene aggiunto un nuovo sensore.

A.4 Assumptions

- Potranno utilizzare il Sistema solo i gestori dei sensori e gli amministratori che hanno delle credenziali d'accesso (Username e Password);
- Il sensore dovrà avere, al momento della sua prima configurazione, un backup di default ed un range di valori accettabili inerenti alle variabili ambientali.
- Il sensore acquisisce i valori ambientali in un lasso di tempo determinato.
- I sensori non devono comunicare tra loro.
- Il segnale verrà eliminato programmaticamente ogni 6 ore.
- Esiste almeno un Amministratore al momento della creazione del Sistema Software.
- Il sistema dovrà supportare lo storage and il processing di almeno 150.000 messaggi al minuto.
- Ogni Utente, quindi Amministratore e Gestore, dovrà avere una postazione di lavoro. In questo modo, nel momento in cui un generico utente riscontrerà una problematica, non verrà compromesso il lavoro di altri utenti.

A.5 Prioritization

ID	Requisito	Priorità
F.R.1.1	Invia Segnale	Alta
F.R. 1.2	Dashboard Gestore	Alta
F.R. 1.4	Sensori a Rischio	Alta
F.R. 1.6	Backup Parametri Sensore	Alta
F.R. 1.9	Dashboard Amministratore	Alta
F.R. 1.3	Monitorare	Media
F.R. 1.5	Ripristino Parametri Sensori	Media
F.R. 1.12	Recupero Password Admin	Media
F.R. 1.13	Risposta Ticket	Media
F.R. 1.7	Aggiungi Sensore	Bassa

F.R. 1.8	Invio Ticket	Bassa
F.R. 1.10	Registrazione Nuovo Admin	Bassa
F.R. 1.11	Registrazione Gestore Sensori	Bassa

B. Software Architecture

C.1 The static view of the system: Component Diagram

Il team ha deciso di utilizzare **Draw.io** per modellare il Component Diagram:

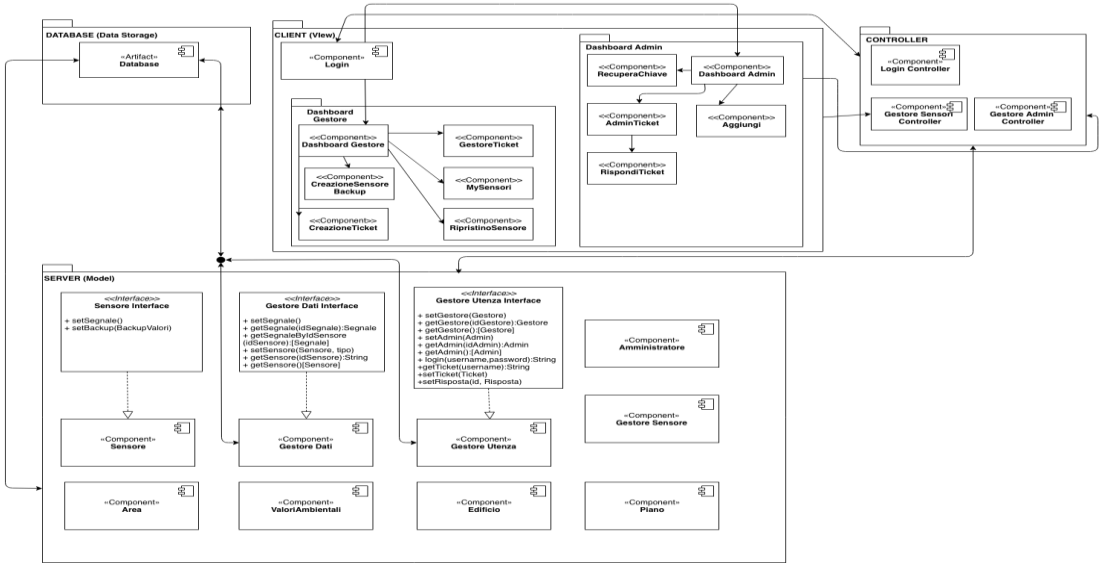


Fig. 18: Component Diagram Monitoraggio Ambientale

Il team ha deciso di utilizzare ***MVC Pattern (Model-View-Controller)***, ovvero un Pattern Architetturale molto diffuso nello sviluppo software, in particolare nell'ambito della programmazione orientata ad oggetti. La suddivisone in strati facilita la scalabilità e la manutenzione del software.

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- ***Model*** fornisce i metodi per accedere ai dati utili all'applicazione;
- ***View*** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- ***Controller*** riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Le Macro-componenti che suddividono la nostra architettura, sono le seguenti:

- **Il DATABASE:** Il sistema dovrà includere *"database"*, ovvero una componente che ha il compito di immagazzinare i dati.
- **CLIENT (GUI):** conterrà altri due package chiamati rispettivamente: *"Dashboard Gestore"* e *"Dashboard Admin"*, che al suo interno avranno come componenti: *"Dashboard Gestore"*, *"CreazioneSensoreBackup"*, *"CreazioneTicket"*,

"GestoreTicket","MySensori","RipristinoSensore""Dashboard Admin", "Aggiungi", "*AdminTicket*", "*RispondiTicket*" e "*RecuperaChiave"*. Questo macro blocco ha il compito di interfacciare l'utente con il sistema (***View***). "*Dashboard Admin*" rappresenta l'interfaccia grafica tra l'amministratore del sistema ed il sistema stesso mentre la "*Dashboard Gestore*" rappresenta l'interfaccia grafica tra il gestore dei sensori ed il sistema del monitoraggio ambientale. Il component "*login*" svolge una funzione importante, ovvero ha il compito di identificare un determinato utente che effettua l'accesso in modo tale da garantire una protezione da attacchi esterni, accidentali o intenzionali (***requirement non functional security***).

- ***CONTROLLER***: è una macro componente che ha il compito di elaborare le richieste in ingresso, gestire gli input e le interazioni del singolo utente ed eseguire la logica del sistema appropriato. Al suo interno troveremo tre componenti, denominate: "*login controller*", "*gestore sensori controller*" e "*gestore admin controller*".
- **SERVER**: è un macro componente che fornisce i metodi per accedere ai dati utili all'applicazione (***Model***). Il ***Model*** non si occupa soltanto dell'accesso fisico ai dati ma anche di creare il necessario livello di astrazione tra il formato in cui i dati sono memorizzati ed il formato in cui i livelli di ***controller e view*** si aspettano di riceverli; oltretutto fornisce una interpretazione intermedia dei dati arricchendo il database con nuove informazioni. Cosa importante, il ***Model*** non contiene direttamente i dati del database, ma ha solo il compito di fornire metodi e di restituire i dati al *controller.*

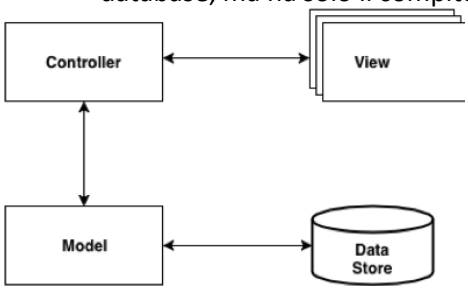


Fig. 19: Schema del Pattern MVC

*L*a macro-componente **SERVER** conterrà i seguenti components: "*Sensore*": E' una classe che verrà implementata dalla component "*Interfaccia Sensore*" che al suo interno conterrà i seguenti metodi: "*setSegnale()*","*setBackup()*" che avranno il compito di scrivere all'interno del macro-componente Database; Il component "*Gestore Dati*" è uno tra i component più importanti, implementato dall'interfaccia "Gestore*Dati*", contenente tutti i metodi attinenti alle funzionalità del sistema, ad esclusione di quelle di utenza. Esso avrà inoltre il compito di leggere e scrivere all'interno del macro-componente "**DATABASE**". Il component "Gestore*Utenza*", sarà implementato dall'interfaccia "*Gestore Utenza*", che contiene tutti i metodi attinenti alle funzionalità di utenza (*setGestore, getGestore, setAdmin, getAdmin, login(username, password), getGestore(Gestore), getAdmin(Admin),getTicket(username),setTicket(Ticket),setRisposta*) e, avrà il compito di leggere e scrivere all'interno del macro-componente "**DATABASE**". Queste tre classi (*"Sensore", "Gestori Dati", "Gestore Utenza"*) avranno anche il compito di garantire il *Requirement Non Functional Scalability*. I components "*Amministratore*","*Area*","*ValoriAmbientali*", "*Edificio","Piano",* rappresentano gli oggetti che verranno utilizzati nel ***Model***.

C.2 The dynamic view of the software architecture: Sequence Diagram

Il team ha deciso di utilizzare **MagicDraw** per modellare il Sequence Diagram:

SEQUENCE DIAGRAM SCENARIO 2: INVIO SEGNALE

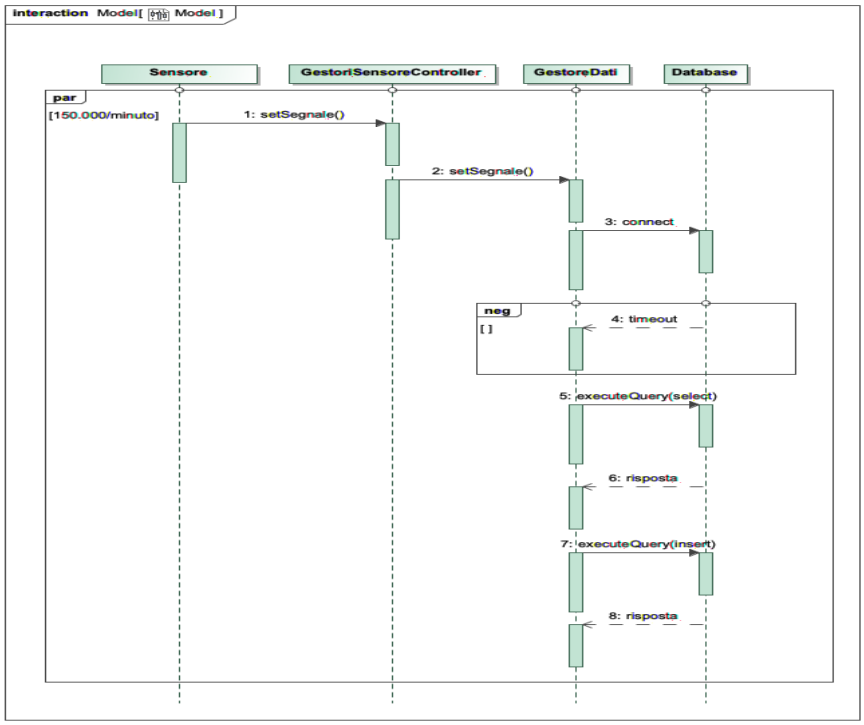


Fig. 20: Sequence Diagram inerente allo Scenario 2: INVIO SEGNALE

Verranno inviati 150.000 segnali in parallelo da parte dei sensori al minuto. Nello specifico, il **Sensore** chiamerà il metodo *"setSegnale()"* al **GestoreSensoriController** che provvederà ad interfacciare la chiamata *setSegnale()* al **GestoreDati**. Il **GestoreDat**i effettuerà la connessione al **Database MonitoraggioAmbientale** mediante il metodo *connect()* e, in caso di mancata connessione, verrà inviato un messaggio di *timeout.* Proseguendo, verrà eseguita una *query che seleziona i dati dei valori ambientali* del sensore nel tempo t in cui è stato effettuato la chiamata al metodo *setSegnale().* Al termine della query mediante la *risposta di successo* da parte del **database,** verrà eseguita una query di insert nella tabella segnale delle variabili ambientali recuperate dalla select precedente.

SEQUENCE DIAGRAM SCENARIO 3: RIPRISTINO

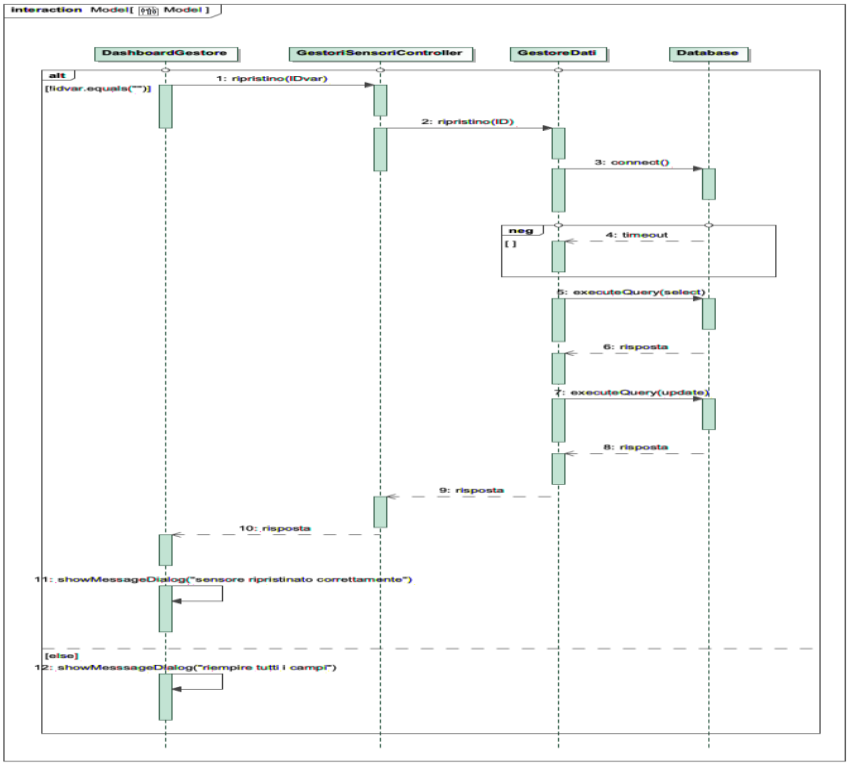


Fig. 21: Sequence Diagram inerente allo Scenario 3: RIPRISTINO

Avremo un *operatore alternative* che verifica se la textbox IDvar è stata riempita, se non viene passato nessuno sensore, verrà effettuata una chiamata al metodo *showMessageDialog* dalla **DashboardGestore** a **sé stesso** che indicherà di riempire il campo. Altrimenti, verrà chiamato il metodo *ripristino(IDvar)* passando come parametro l'id del sensore al **GestoreSensoriControlle**r che lo interfaccerà al **GestoreDati.** Il **GestoreDat**i effettuerà la connessione al **Database MonitoraggioAmbientale** mediante il metodo *connect()* e, in caso di mancata connessione, verrà inviato un messaggio di *timeout.* Proseguendo, verrà eseguita una *query che seleziona* i valori ambientali ottimali da utilizzare per effettuare il ripristino. Al termine della query mediante la *risposta di successo* da parte del **database,** verrà eseguita una query di update sulla tabella sensore che modificherà i valori ambientali presenti nel sensore con quelli ottimali. Al termine, il **GestoreDati** invierà un *messaggio di risposta* con avvenuta modifica al **GestoreSensoriController** che lo interfaccerà alla **DashboardGestore.** La **DashboardGestore** effettuerà una chiamata al metodo *showMessageDialog* a **sé stesso** che indicherà il corretto ripristino del sensore.

C. ER Design

Il team ha deciso di utilizzare **draw.io** per la realizzazione dell'ER-Model:

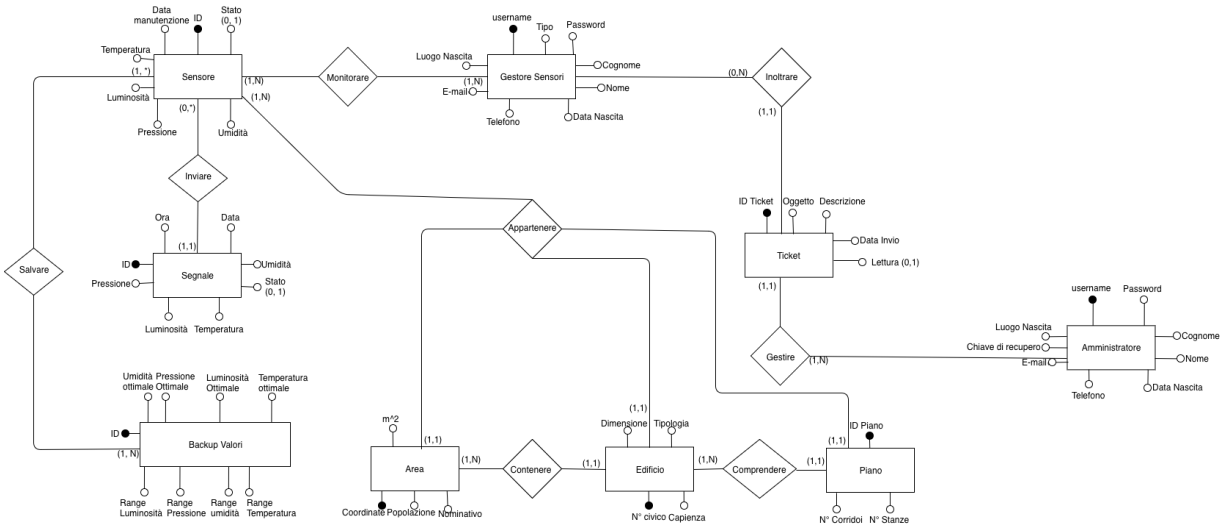


Fig. 22: Modello ER Monitoraggio Ambientale

Modello Relazionale:

Sensore (**ID**, DataManutenzione, Stato, Temperatura, Luminosità, Pressione, Umidità, *Gestore, Piano, Edificio, Area*).

Segnale (**ID**, Data, Ora, Stato, Umidità, Temperatura, Luminosità, Pressione, *IDSensore*).

BackupValori (**ID**, RangeLuminosità, RangePressione, RangeUmidità, RangeTemperatura, UmiditàOttimale, PressioneOttimale, TemperaturaOttimale, LuminositàOttimale, *Sensore*).

GestoreSensori (**Username**, Password, Nome, Cognome, Tipo, LuogoNascita, DataNascita, Email, Telefono).

Area (**Coordinate**, Popolazione, Nominativo, m^2).

Edificio (**NCivico**, Capienza, Tipologia, Dimensione, *Area*).

Piano (**IDPiano**, NCorridoi, NStanze, *Edificio*).

Ticket (**IDTicket**, Oggetto, Descrizione, DataInvio, Lettura, *Mittente, Destinatario*).

Amministratore (**Username**, Password, Nome, Cognome, DataNascita, LuogoNascita, Telefono, Email, ChiaveDiRecupero).

DESCRIZIONE E-R MODEL

1) Consideriamo innanzitutto l'entità **SEGNALE**. Tale entità è costituita dagli attributi ***id****, data, ora, stato, umidità, temperatura, luminosità, pressione*. Tra questi, l'***id*** rappresenta l'attributo che permette di contraddistinguere univocamente ciascun segnale, *data* e *ora* rappresentano, appunto, giorno e ora dell'invio di un segnale da parte del sensore associato mediante la ForeignKey *Sensore*, e i rimanenti attributi rappresentano le informazioni ambientali (*umidità, temperatura, luminosità, pressione*) e le informazioni di stato di funzionamento (0,1) che il sensore ha inviato nel momento t = x, dato come detto precedentemente dalla data e ora.

2) L'entità **SENSORE**, è costituita dagli attributi delle variabili ambientali che vengono passati al segnale ad esso associato e dovrà occuparsi quindi di inviarli periodicamente. Un sensore potrà inviare (0,*) segnali: questa cardinalità sta a significare che un sensore è in grado di inviare indefiniti segnali oppure nessuno nel caso di un'anomalia. L'attributo *data manutenzione*, rappresenta appunto un'eventuale manutenzione che è stata effettuata sul sensore. Chiaramente ciascun **SENSORE** ha un'**id** (caratterizzato da un codice numerico auto incrementale) come chiave primaria che permette di distinguere univocamente il sensore rispetto ad altri.

3) Tale entità deve occuparsi anche di salvare le informazioni che le vengono passate per verificare in seguito se rientrano in parametri accettabili e veritieri, e per permettere al Gestore dei Sensori di effettuare il ripristino delle variabili ambientali attinenti ad esso. Tutto questo rappresentato dall'associazione tra, appunto, SENSORE e l'entità **BACKUP VALORI**. Nello specifico i sensori possono salvare, cioè backuppare, uno o indefiniti valori, ovvero con cardinalità (1,*). Tornando all'entità BACKUP VALORI, costituita, quindi, dagli attributi *umidità ottimale, temperatura ottimale, luminosità ottimale, pressione ottimale* utilizzati dal gestore sensori per permettere il ripristino delle variabili ambientali del sensore nel caso in cui essi risultano fuori soglia, e i corrispondenti range di valori (valori che riportano il sensore fuori soglia in caso in cui essi vengano superati).

I sensori sono distribuiti all'interno di zone geografiche, ovvero grazie ad essi si potrà monitorare un'intera area, gli edifici che la compongono e infine i piani che appartengono a ciascun edificio. In dettaglio uno o più sensori appartengono a ciascun piano, edificio, area identificate dall'entità **PIANO, EDIFICIO, AREA.**

4) L'entità **PIANO** presenta i seguenti attributi: l'***id piano,*** *N. stanze, N. corridoi.* L'id (rappresentato da un codice numerico) serve per contraddistinguere i vari piani e gli altri due attributi descrivono appunto il numero di stanze e di corridoi che formano ciascun piano.

5) Detto ciò, dall'associazione tra l'entità **PIANO** e l'entità **EDIFICIO** si è in grado di monitorare in dettaglio tutti gli edifici. Ciascun Edificio potrà avere più piani (cardinalità 1,N) e ciascun piano descriverà quindi una parte di esso (cardinalità 1,1). Gli attributi dell'entità EDIFICIO sono ***Ncivico****, capienza, dimensione, tipologia.* Il N° civico rappresenta la chiave primaria (rappresentato da un numero), *dimensione, capienza* e *tipologia* forniscono informazioni dei vari edifici, quindi se sono ospedali, biblioteche, ecc... . Da tutti questi dati si potra'

distinguere piani che ad esempio hanno codici uguali ma comunque appartengono ad edifici diversi.

6) I vari edifici poi sono contenuti in aree. Ovvero, associazione **EDIFICO** e **AREA**. Anche qui, ogni area conterrà uno o più edifici (cardinalità 1,N) e ciascun edificio descriverà ogni parte di una zona (cardinalità 1,1). L'entità AREA ha come attributi secondari *nominativo, npopolazione, metri quadrati* e come attributo principale (chiave primaria) le **coordinate (**rappresentate da una stringa**)** di dove si trova.

7) Passiamo ora all'entità **GESTORE DEI SENSORI**. Il gestore ha i seguenti attributi: ***username****, password, tipo, cognome, nome, datanascita, luogonascita, telefono, e-mail*. In dettaglio, l'attributo ***username*** identifica univocamente tutti i gestori e rappresenta con la *password* le credenziali per accedere alla propria area riservata. Mentre l'attributo *tipo* rappresenta la categoria a cui appartengono i gestori: gestore *città*, gestore *area*, gestore *edificio*. I vari gestori devono monitorare la zona d'interesse di loro competenza e quindi vedere in modo semplice ed accurato e con il giusto livello di dettaglio le informazioni rilevanti. Queste due operazioni sono rappresentate dall'associazione "monitorare" tra entità SENSORE ed entità GESTORE DEI SENSORI con cardinalità (1,N) per rappresentare il fatto che i gestori possono monitorare più sensori; e la possibilità da parte dei gestori di eseguire un'operazione di zoom-in,zoom-out sulla zona di loro competenza.

8) I gestori non possono comunicare tra di loro. Se si verifica un malfunzionamento di qualunque tipo, il gestore deve comunicarlo agli amministratori del sistema, tramite opportuni **TICKET**. Questa azione e' rappresentata dall'associazione "inoltrare" tra entità **GESTORE DEI SENSORI** e **TICKET**. Associazione con cardinalità (0,N) per descrivere che i gestori possono inviare più di un ticket e con cardinalità (1,1) per rappresentare il fatto che lo stesso ticket non potrà essere inoltrato da gestori diversi.

9) Infine troviamo l'entità **AMMINISTRATORE**, ovvero coloro che devono garantire il corretto funzionamento del sistema. Caratterizzato dai seguenti attributi: ***username****, password, cognome, nome, datanascita, luogonascita, telefono, e-mail, chiavedirecupero*. La chiave primaria e' rappresentata dall'*username*. L'attributo chiave di recupero, specifica che nel caso in cui un amministratore perde un dato tra username e password sia possibile recuperarlo. Come abbiamo accennato poco prima, l'amministratore deve gestire i ticket che riceve dai vari gestori. Ciò è specificato dall'associazione "gestire" tra entità **TICKET** e **AMMINISTRATORE**. Anche qui le cardinalità (1,1) ed (1,N) rappresentano il fatto che uno stesso ticket non può essere gestito da più amministratori ma comunque un amministratore può ricevere e gestire più di un ticket.

D. Class Diagram of the implemented System

Il team ha deciso di utilizzare **draw.io** per la realizzazione del Class Diagram:

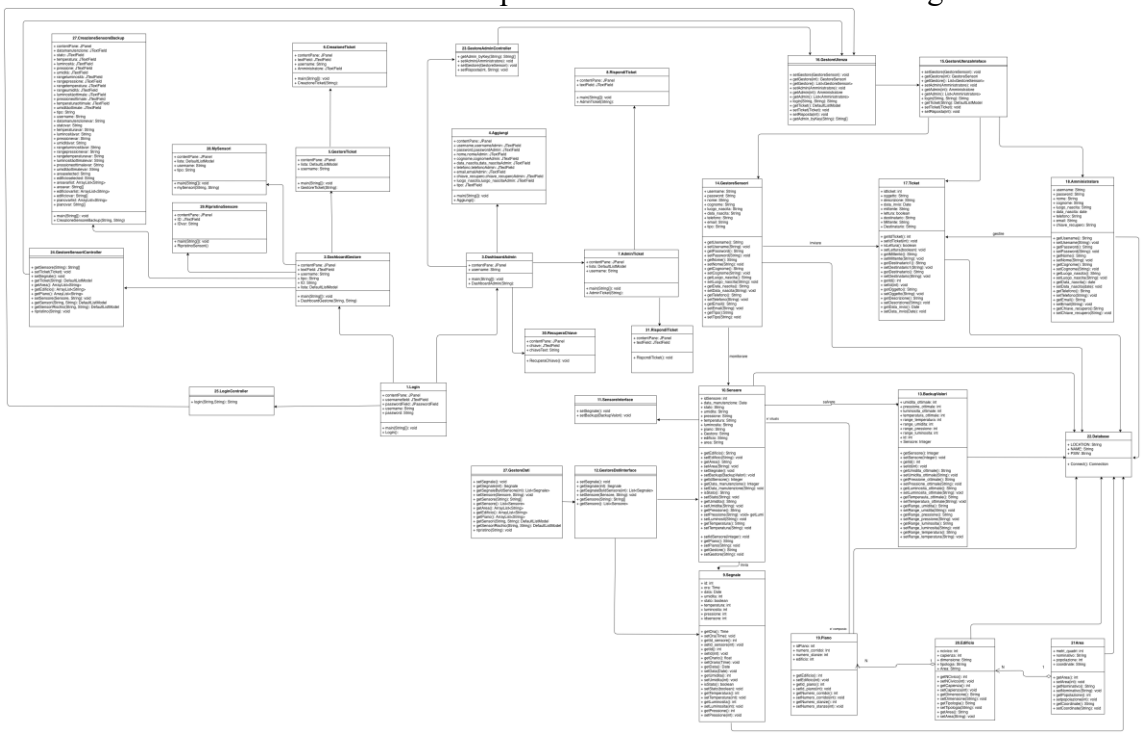


Fig.23: Class Diagram Monitoraggio Ambientale

Il ***Class Diagram*** riporta tutte le classi usate all'interno del nostro sistema software:

1) **LOGIN:** E' una classe che implementa la classe *JFrame*. La classe Login è presente nel file *Login.java* nel package *View* che permette l'accesso alle Dashboard da parte di un Gestore dei Sensori o di un Admin; E' una classe importante perché permette di effettuare il controllo sul tipo di Utente che stà provando ad accedere al sistema.
2) **DASHBOARDGESTORE**: E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *DashboardGestore.java* nel package *View.DashboardGestore* che permette ai Gestore dei Sensori di monitorare i sensori ed inviare dei Ticket; E' una classe importante perché rappresenta l'interfaccia grafica del nostro sistema software.
3) **DASHBOARDADMIN**: E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *DashboardAdmin.java* nel package *View.DashboardAdmin* che permette all'Admin di aggiungere un nuovo Admin/Gestore, gestire i Ticket e recuperare le credenziali di un altro Admin; E' una classe importante perché rappresenta l'interfaccia grafica del nostro sistema software.
4) **AGGIUNGI**: E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *Aggiungi.java* nel package *View.DashboardAdmin* che permette all'Admin di aggiungere un nuovo gestore o un nuovo Admin; Tale classe permette la creazione di un

oggetto di tipo Amministratore e di tipo Gestore dei Sensori. Una volta creati, gli oggetti vengono passati come parametri al *GestoreAdminController*.

5) **GESTORETICKET**: E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *GestoreTicket.java* nel package *View.DashboardGestore* che permette di visualizzare/gestire i Ticket inviati dal Gestore dei Sensori ad un Admin;
6) **CREAZIONETICKET**: E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *CreazioneTicket.java* nel package *View.DashboardGestore* che permette al gestore dei sensori di creare un nuovo Ticket; E' una classe importante perché permette di istanziare un oggetto di tipo Ticket. Una volta creato, l'oggetto di tipo Ticket, viene passato come parametro al *GestoreSensoriController*.
7) **ADMINTICKET:** E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *AdminTicket.java* nel package *View.DashboardAdmin* che permette di listare/gestire tutti i Ticket che i Gestori dei Sensori hanno inviato all'Admin;
8) **RISPONDITICKET**: E' una classe che implementa la classe *JFrame*. E' una classe presente nel file *RispondiTicket.java* nel package *View. NeDashboardAdmin* che permette di rispondere ai Ticket dei Gestori; Nello specifico, viene passato come parametro al *GestoreAdminController* l'id del Ticket che deve essere modificato.
9) **SEGNALE**: E' una classe presente nel file *Segnale.java* nel package *Model.Components* che permette di costruire oggetti di tipo Segnale nel momento in cui viene invocato il metodo ***setSegnale()*** dalla classe Sensore;
10) **SENSORE:** E' una classe presente nel file *Sensore.java* nel package *Model.Components* che permette di effettuare delle operazioni di scrittura/lettura sul Database, nel preciso nella tabella BackupValori e Segnale; E' una classe importante perché senza di essa non sarebbe possibile effettuare operazioni di insert, backup e segnale. L'invio del segnale avviene mediante il metodo ***setSegnale(),*** il backup delle variabili ambientali ottimali ed il range su cui il sensore si deve basare per avvertire il gestore dei sensori di presenza di valori fuori soglia, avvengono mediante il metodo ***setBackup()***;
11) **SENSOREINTERFACE:** E' una classe Interfaccia che verrà implementata dalla classe Sensore, dove sono presenti le dichiarazioni dei metodi ***setSegnale()*** e ***setBackup()***;
12) **GESTOREDATIINTERFACE**: E' una classe Interfaccia che verrà implementata dalla classe GestoreDati, dove sono presenti le dichiarazioni dei metodi di set e get inerenti ai Sensori, ai Segnali e alle rispettive liste.
13) **BACKUPVALORI**: E' una classe presente nel file *BackupValori.java* nel package *Model.Components* che permette di costruire oggetti di tipo *BackupVal*ori.
14) **GESTORESENSORI:** E' una classe presente nel file *GestoreSensori.java* nel package *Model.Components* che permette di costruire oggetti di tipo *GestoreSensori*;
15) **GESTOREUTENZAINTERFACE:** E' una classe Interfaccia che verrà implementata dalla classe *GestoreUtenza*, dove sono presenti le dichiarazioni dei metodi di Get e Set inerenti al Gestore, all'Admin e al Ticket.
16) **GESTOREUTENZA:** E' una classe che implementa la classe GestoreUtenzaInterface dove sono presenti le definizioni dei metodi riepilogati nel punto precedente. E' una classe molto importante dove è presente la logica della Login e della modifica, restituzione e aggiunta dei dati inerenti all'Admin, al Gestore dei Sensori, del Ticket ed infine, è presente la logica per il recupero delle credenziali d'accesso mediante il metodo *getAdmin_byKey(String)* dove verrà passato come parametro la chiave di recupero.
17) **TICKET:** E' una classe presente nel file *Ticket.java* nel package *Model.Components* che permette di costruire oggetti di tipo Ticket.

18) **AMMINISTRATORE:** E' una classe presente nel file *Amministratore.java* nel package *Model.Components* che permette di costruire oggetti di tipo Amministratore.
19) **PIANO:** E' una classe presente nel file *Piano.java* nel package *Model.Components* che permette di costruire oggetti di tipo Piano. L'insieme degli oggetti di tipo Piano rappresentano un ***agglomerato*** dell'oggetto di tipo Edificio.
20) **EDIFICIO:** E' una classe presente nel file *Edificio.java* nel package *Model.Components* che permette di costruire oggetti di tipo Edificio. L'insieme degli oggetti di tipo Edificio rappresentano un ***agglomerato*** dell'oggetto di tipo Area.
21) **AREA:** E' una classe presente nel file *Area.java* nel package *Model.Components* che permette di costruire oggetti di tipo Area;
22) **DATABASE:** Rappresenta la classe situata nel file *Database.java* presente nel package *Model.DB* che permette lo storage dei dati, infatti sono presenti metodi come ***Connect()*** con l'obiettivo di instaurare una connessione tra il database e il programma in Java;
23) **GESTOREADMINCONTROLLER**: E' una classe presente nel file *GestoreAdminController.java* nel package *Controller* che permette di interfacciare le chiamate dei metodi effettuate dalla View mediante, nello specifico, l'utilizzo della chiamata al metodo, che avrà il suo stesso nome, e che sarà presente nel *Model.Components;*
24) **GESTORESENSORICONTROLLER**: E' una classe presente nel file *GestoreSensoriController.java* nel package *Controller* che permette di interfacciare le chiamate dei metodi effettuate dalla View mediante, nello specifico, l'utilizzo della chiamata al metodo, che avrà il suo stesso nome, e che sarà presente nel *Model.Components;*
25) LOGINCONTROLLER: E' una classe presente nel file *LoginController.java* nel package *Controller* che permette di leggere i dati passati dalla login nel *Model* e restituire una stringa contenente *l'username e password* dell'utente che accede.
26) **GESTOREDATI**: E' una classe che implementa la classe *GestoreDatiInterface* dove sono presenti le definizioni dei metodi riepilogati nel *punto 12)*. E' una classe molto importante dove è presente la logica della modifica, restituzione ed aggiunta dei Sensori e dei Segnali ed infine, è presente la logica per il ripristino di un Sensore mediante il metodo *ripristino(String)* dove verrà passato come parametro l'Id del Sensore da ripristinare.
27) CREAZIONESENSOREBACKUP: E' una classe che implementa la classe JFrame. E' una classe presente nel file *CreazioneSensoreBackup.java* del package *View.DashboardGestore* che permette al Gestore dei Sensori di creare un nuovo sensore con il rispettivo Backup. E' una classe importante perché permette di instanziare un oggetto di tipo Sensore e successivamente un oggetto di tipo BackupValori.
28) **MYSENSORI:** E' una classe che implementa la classe JFrame. E' una classe presente nel file *MySensori.java* nel package View.DashboardGestore che permette al Gestore dei Sensori di listare tutti i sensori di sua competenza mediante l'utilizzo di una JList chiamata lista;
29) **RIPRISTINOSENSORE:** E' una classe che implementa la classe JFrame. E' una classe presente nel file *RipristinoSensore.java* nel package View.DashboardGestore che permette al Gestore dei Sensori di ripristinare un sensore passato dalla textbox nominata ID;
30) **RECUPERACHIAVE:** E' una classe che implementa la classe JFrame. E' una classe presente nel file RecuperaChiave.java nel package View.DashboardAdmin che permette

all'Amministratore di recuperare le credenziali d'accesso di un altro Admin passando nella textbox denominata "chiave", la chiave di recupero.

31) **RISPONDITICKET:** E' una classe che implementa la classe JFrame. E' una classe presente nel file RispondiTicket.java nel package *View.DashboardAdmin* che permette all'Amministratore di rispondere ad un determinato Ticket inviato da un Gestore dei Sensori.

E. Design Decisions

1) DECISION CLIENT-SIDE: Il team ha deciso per lo sviluppo del software Monitoraggio Ambientale di creare in particolare una Desktop App. Tale decisione è stata concordata perché si è ritenuto opportuno avere un ambiente software sulla propria macchina di lavoro. Inoltre abbiamo pensato che, in futuro grazie alla progettazione del team, questo sistema software potrebbe essere esteso anche per altre piattaforme (ad esempio Web) grazie alla scalabilità utilizzata.

2) **DECISION MVC PATTERN:** il team ha deciso di impiegare il "pattern MVC" per rappresentare l'architettura del software e il component diagram ad esso collegato. Tale decisione e' stata concordata in primo luogo per organizzare il software in modo ottimale, in secondo luogo perche' grazie a questa tipologia di pattern possiamo organizzare il codice in modo piu' logico, dividendolo quindi in tre parti. Siamo cosi' in grado di facilitare la **scalabilita'** e la **manutenzione** dell'applicazione.

3) **ARCHITTETURA DATABASE:** Il team da deciso di utilizzare, per l'implementazione del database **monitoraggioambientale,** il **linguaggio d'interrogazione MySQL** che permette di sviluppare database che si basano sui modelli relazionali e sull'**engine INNODB**. Il **modello Relazionale** viene considerato attualmente il modello più semplice ed efficace, perché è più vicino al modo consueto di pensare i dati, e si adatta in modo naturale alla classificazione e alla strutturazione dei dati. Il modello relazionale è più intuitivo e più espressivo per la strutturazione dei dati, rispetto agli altri modelli come quello gerarchico e reticolare. E' stato deciso di adottare l'engine INNODB rispetto l'engine MyISAM perché è molto più rigido nell'integrità dei dati e nella referenzialità tra le relazioni mediante le ForeignKey, ed è molto più veloce nella fase di scrittura e lettura sulle tabelle visto che i dati presenti nelle relazioni Segnale e BackupValori saranno modificati in un lasso di tempo molto piccolo.

4) **PERFORMANCE: -** ENGINE INNODB: Insert 150.000 tuple: 0,14 ms x 150.000 segnali = 21 secondi; - ENGINE MYISAM: Insert 150.000 tuple: 0,18 ms x 150.000 segnali = 27 secondi. ***SCELTA****: I*l team ha deciso di utilizzare l'engine InnoDB visto che permette di scrivere 150.000 segnali al minuto con soli 21 secondi a differenza di 27 secondi di MyIsam.

5) **SCELTA DEL LINGUAGGIO DI PROGRAMMAZIONE**: Il team, decidendo di sviluppare una Desktop App, ha scelto di programmare il software mediante il linguaggio Java rispetto a C++. I motivi sono i seguenti: Java consente di fare facilmente ciò che in C++ è difficile e costoso. Ad esempio, con sole poche righe di codice si possono creare più thread di esecuzione. Atra motivazione: Java è un linguaggio indipendente alla piattaforma, in quanto la compilazione delle sorgenti avviene in codice, denominato bytecode, diverso dal linguaggio macchina, ovvero come riporta il famoso slogan creato da "Sun MicroSystems": "*write once, run everywhere*".

F. Explain how the FRs and the NFRs are satisfied by design

- ***REQUISITI FUNZIONALI:***
  1. **Dashboard (implementato)**: Dopo aver effettuato l'accesso dalla GUI della Login, a seconda dell'utente (Amministratore o Gestore dei Sensori) verrà visualizzata la Dashboard opportuna. **(Vedere GUI Requirements Fig.re 3 e 10)**;
  2. **Invio Segnale (Implementato):** Utilizzando il metodo setSegnale() presente nella classe GestoreDati nel file GestoreDati.java. Nello specifico verrà effettuata una connessione al Database, si eseguirà una query che permette di recuperare i dati dei valori ambientali e lo stato di funzionamento dell'oggetto di tipo Sensore che ha richiamato il metodo. Questi dati saranno salvati all'interno di un array di stringhe che verrà utilizzato per effettuare la query di insert nella table segnale. Ovviamente, questo metodo verrà richiamato da ogni sensore allo scadere del minuto. Per permettere ai sensori di inviare periodicamente i segnali con le varie informazioni ambientali, viene generato un Thread per l'invio del segnale ogni minuto al momento del lancio dell'applicazione, nello specifico, nel lancio della Dashboard GestoreSensori nel file DashboardGestore.java situato nel package View.DashboardGestore;
  3. **Backup Parametri Sensore (implementato)**: Utilizzando il metodo setBackup(BackupValori b) presente nella Classe Sensore nel file Sensore.java. Verrà effettuata una connessione al Database, si eseguirà una query che farà una insert dei valori che sono stati passati dalle textbox della View CreazioneSensoreBackup.java, ovviamente verrà rispettato l'utilizzo del Pattern MVC;
  4. **Ripristino Parametri Sensore (implementato):** utilizzando il metodo ripristino(String ID) verrà creata la connessione al Database, verrà eseguita la query per recuperare i valori ottimali inerenti al Sensore passato dalla textbox, dopo di che verrà effettuata una query di update che permetterà di sostituire i valori ambientali attuali con quelli ottimali recuperati dalla query precedente;
  5. **Interazione (Implementato):** A seconda della Dashboard che l'utente sta utilizzando, sarà possibile instaurare una comunicazione tra Amministratore e Gestore. Nel caso dell'Amministratore, andando nella sezione "Ticket", potrà visualizzare la lista dei ticket e rispondere ad un determinato messaggio inviatogli da un Gestore. **(Vedi GUI Requirements Fig.ra 4).** Viceversa, nel caso del Gestore dei Sensori, andando nella sezione "Ticket", potrà inviare un ticket per informare gli Amministratori di un determinata problematica. **(Vedi GUI Requirements Fig.re 15 e 16).** Nel momento in cui un Amministratore

risponderà al ticket, tutti gli altri non potranno rispondere a quel determinato messaggio. Questo perché ogni ticket ha un campo lettura con valore booleano che viene settato ad 1 nel momento in cui un determinato Amministratore lo leggerà;

6. **Registrazione Gestore Sensori e Admin (Implementato):** Tramite la Dashboard Admin, gli Amministratori potranno inserire uno o più Gestori dei Sensori oppure un nuovo Admin assegnandogli le specifiche credenziali d'accesso. **(Vedi GUI Requirements Fig. 5).** Nello specifico cliccando sul bottone "Aggiungi Gestore" o "Aggiungi Admin". Mediante i metodi setAdmin(Amministratore a) setGestore(GestoreSensori g) presenti nel file GestoreUtenza.java verrà creata una connessione al Database, verrà effettuata una query insert dove sarà aggiunto il Gestore dei Sensori appropriato o un Amministratore passato dalle textbox presenti nella View Agiungi.java;
7. **Recupera chiave (Implementato):** Mediante il metodo getAdmin_byKey(String chiave) verrà creata una connessione al database, verrà effettuta una query di select che restituirà username e password dell'Amministratore che ha bisogno di recuperare una delle due credenziali. Sarà creato un array di Stringhe che conterrà come primo elemento l'username, come secondo elemento la password; per permettere al Controller di, appunto, restituire i dati cercati alla View**.**

- ***REQUISITI NON FUNZIONALI:***
  1. **DEPENDABILITY:**
     - **FAULT TOLERANCE (implementato):** *Il sistema deve continuare ad offrire i servizi anche se si è persa la funzionalità di una componente.* Questo vincolo viene soddisfatto, dal punto di vista implementativo, istanziando, ovvero effettuando operazioni di scrittura nella relazione Sensore di almeno due sensori per piano;
     - **SECURITY (Implementato)***: Il sistema deve essere in grado di proteggersi da attacchi esterni, accidentali o intenzionali.* Questo vincolo viene soddisfatto attraverso l'interfaccia Login dove ciascun utente deve immettere le proprie credenziali (Username e Password) per poter accedere al Sistema. Al momento del click sul bottone "Login" **(Vedi GUI Requirements Fig.re 1 e 2)**, verrà effettuato un controllo sul Tipo per verificare per prima cosa se è un Amministratore o un Gestore, nel momento del click sul bottone Login e, nel caso in cui risulti un Gestore, verrà effettuato un controllo sul 'Tipo' di Gestore (Edificio, Area, Città);
  2. **SCALABILITY (implementato):** *Il sistema garantisce un'architettura scalabile per supportare future espansioni.* Questo vincolo è stato soddisfatto attraverso l'utilizzo dei bottoni 'Aggiungi Sensore' ed 'Aggiungi Admin/Gestore' nelle corrispondenti Dashboard Admin e Dashboard Gestore Sensori **(Vedi GUI Requirements Fig.re 8 e 9)**. Per una descrizione più

approfondita dei metodi implementati leggere la sezione Requisiti Funzionali punto 3 e punto 6(nella sezione Mapping).

3. **USABILITY (Implementato):** *Il sistema deve offrire una User-Experiencie-Friendly*. Questo vincolo verrà soddisfatto dalle Dashboard di ciascun utente che può utilizzare il Sistema. Nello specifico mediante l'utilizzo di "messages dialog" presenti nelle View, mediante una disposizione ordinata e intuitiva di bottoni e textbox. **(Vedi GUI Requirements)**
4. **PERFORMANCE (Implementato):** *Il sistema deve risultare efficiente e deve lavorare con tempi d'esecuzione accettabili.* Il team ha effettuato i test in termini di Tempi d'esecuzione nel caso di una insert di segnale da parte di un sensore ed Occupazione della Memoria del database, nello specifico **vedere Decision Design 4),** mentre per l'occupazione di memoria, utilizzando un engine InnoDB, il database occupa più spazio rispetto ad un database che utilizza l'engine MyISAM, ma per avere dei tempi d'esecuzione molto più perfomanti, abbiamo deciso di adottare l'engine INNODB

G. Effort Recording

PERT

Il team ha deciso di utilizzare ***ProjectLibre*** per modellare il Pert Chart:

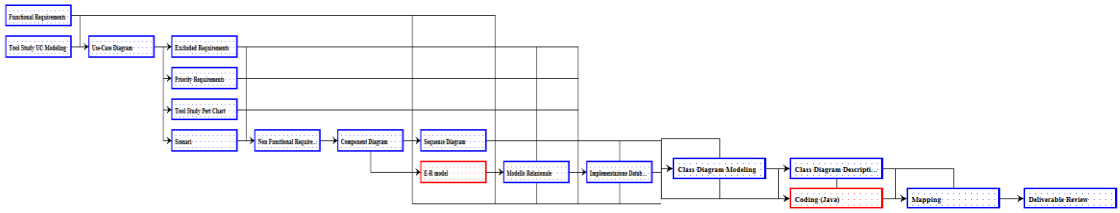


Fig. 24: Pert Charts Monitoraggio Ambientale

Summary Statistics

TASK	*DOING*	*LEARNING*
Use-cases Review	9 H	
Coding Java Prototype	31 H	6 H 10 MIN
GUI Requirements	5 H	
Deliverable Review	14 H 40 MIN	
Component Diagram Modeling and Description	6 H	
Sequence Diagram Modeling and Description	3 H	
Class Diagram Modeling and Description	13 H 30 MIN	
Detail Scenarios	4 H	
ER Model Review	3 H	
Explain how the FRs and the NFRs are satisfied by design	3 H	
Pert Charts Modeling	1 H 20 MIN	
Appendix.Prototype	2 H	
TOTALE	*95 H 30 MIN*	*6 H 10 MIN*

Appendix. Prototype

Il sistema per poter funzionare ha bisogno dei seguenti programmi:

- - MySQL Community Server: (solo utenti MacOS) https://dev.mysql.com/downloads/mysql/
- - MySQL Workbench: (solo utenti MacOS) https://dev.mysql.com/downloads/workbench/

- Java SE Development Kit 8: (per tutti gli utenti)
  https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- MySQL Installer: (solo utenti Windows)
  https://dev.mysql.com/downloads/windows/installer/8.0.html

Per poter usare il nostro sistema:

Per sistema operativo MacOS:

- Avviare MySQL Community Server tramite il menù "preferenze di sistema" su sistema cliccando sull'icona del programma e successivamente su "start my sql server". Inserire la password scelta in fase di installazione (consigliamo "root" per entrambi i campi) ed accertarsi del corretto funzionamento del Server tramite le spie verdi.

- **Avviare successivamente MySQL Workbench ed eseguire una scansione dei server disponibili, verrà trovato il server MySQL precedentemente installato. Cliccare sul server ed inserire le password scelte per eseguire la connessione.**
  Nella seconda schermata cliccare su:
  Data Import -> import from self-contained file -> tramite il menù di ricerca selezionare il server "monitoraggioambientale.sql" -> infine cliccare su "start import" in basso a destra della schermata.
  Una volta effettuato con successo queste operazioni sarà possibile visualizzare il nostro server "monitoraggio ambientale" sulla sinistra della finestra corrente.

- Avviare il terminale ed eseguire la riga di comando "java -jar softing.jar" per lanciare il programma che deve trovarsi nella stessa Path di esecuzione del comando.
  Verrà mostrata una finestra di login, da qui è possibile inserire le credenziali per accedere alla Dashboard di appartenenza.

Su sistema operativo Windows:

1. Scaricare ad avviare l'installer di MySQL V.5.7 e seguire la procedura guidata per installare sia il server che il Workbench sulla propria macchina, quando richiesto selezionare "Developer Default"
   1.1. Una volta pronto verrà chiesto di configurare i prodotti appena installati, scegliere l'impostazione "Standalone MySQL Server"
   1.2. Successivamente nella finestra "Type and Networking" passare oltre senza modificare alcuna impostazione
   1.3. Scegliere infine delle credenziali (inserire come password "root") che verranno usate dall'applicativo ad ogni accesso
   1.4. È possibile lasciare le successive impostazioni di default

- **Se tutto è andato a buon fine sarà possibile accedere al database tramite le stesse modalità descritte precedentemente per MacOS.**
- Mantenendo MySQL Workbench V.6.3CE in esecuzione basterà eseguire con un doppio click sull'applicativo softing.jar per lanciare il programma.

Descrizione sistema:

Una volta avviato il software ci si troverà davanti una pagina di login, qui sarà possibile accedere alla propria dashboard inserendo le credenziali corrette:

Come Amministratore: Username: Muccini23 Password: muc123

Come Gestore dei sensori (Città): Username: Salvatore96 Password: sal96

Come Gestore dei sensori (Area): Username: Tobia56 Password: tob56

Come Gestore dei sensori (Edificio): Username: Fabrizio96 Password: fab96

FR Coding:

1) Invio Segnale

*GestoreDati.java

```
20  @Override
21  public void setSegnale() {
22     String[] a = new String[5];
23     Connection c=new Database().Connect();
24     Statement st;
25     Statement st2;
26     try {
27        st = c.createStatement();
28        ResultSet rs = st.executeQuery("select Stato, Luminosità, Pressione, Temperatura, Umidità from sensore");
29        while(rs.next()) {
30           a[0]=rs.getString("Stato");
31           a[1]=rs.getString("Luminosità");
32           a[2]=rs.getString("Pressione");
33           a[3]=rs.getString("Temperatura");
34           a[4]=rs.getString("Umidità");
35        }
36     } catch (SQLException e) {
37        // TODO Auto-generated catch block
38        e.printStackTrace();
39     }
```

*GestoreDati.java

```
40      try {
41          st2 = c.createStatement();
42          st2.executeQuery("insert into segnale(Stato, Temperatura, Luminosità, Pressione, Umidità) values "
43                  + "('"+a[0]+"','"+a[3]+"','"+a[1]+"','"+a[2]+"','"+a[4]+"' ");
44      } catch (SQLException e) {
45          // TODO Auto-generated catch block
46          e.printStackTrace();
47      }
48
49  }
50
```

2) Backup di un Sensore

*GestoreDati.java

```
69  @Override
70  public void setSensore(Sensore s,String tipo) {
71      // TODO Auto-generated method stub
72      //Database
73
74          //Connessione al Db
75          Connection c=new Database().Connect();
76          Statement st;
77          //Prende la connessione
78              try {
79                  if(tipo.equals("Edificio")) {
80                      st = c.createStatement();
81                      String tot="insert into sensore(DataManutenzione, Stato, Temperatura, "
82                              + "Luminosità, Pressione, Umidità, Gestore, Piano, Edificio, Area) values"
83                              + "('"+s.getData_manutenzione()+"','"+s.isStato()+"','"+s.getTemperatura()+"','"+
84                              s.getLuminosita()+"','"+s.getPressione()+"','"+s.getUmidita()+"','"+s.getGestore()+"','"+
85                              s.getPiano()+"','"+s.getEdificio()+"','"+s.getArea()+"')";
86                      st.executeUpdate(tot);
87                  }
```

*GestoreDati.java

```
88                  if(tipo.equals("Area")) {
89                      st = c.createStatement();
90                      String tot="insert into sensore(DataManutenzione, Stato, Temperatura, Luminosità, Pressione, "
91                              + "Umidità, Gestore, Edificio, Area) values('"+s.getData_manutenzione()+"','"+s.isStato()+"','"+
92                              s.getTemperatura()+"','"+s.getLuminosita()+"','"+s.getPressione()+"','"+s.getUmidita()+"','"+
93                              s.getGestore()+"','"+s.getEdificio()+"','"+s.getArea()+"')";
94                      st.executeUpdate(tot);
95                  }
96                  if(tipo.equals("Città")) {
97                      st = c.createStatement();
98                      String tot="insert into sensore(DataManutenzione, Stato, Temperatura, Luminosità, Pressione, "
99                              + "Umidità, Gestore, Area) values('"+s.getData_manutenzione()+"','"+s.isStato()+"','"+
100                             s.getTemperatura()+"','"+s.getLuminosita()+"','"+s.getPressione()+"','"+s.getUmidita()+"','"+
101                             s.getGestore()+"','"+s.getArea()+"')";
102                     st.executeUpdate(tot);
103                 }
104             } catch (SQLException e) {
105                 // TODO Auto-generated catch block
106                 e.printStackTrace();
107             }
108 }
```

*Sensore.java

```
@Override
public void setBackup(BackupValori b) {
   // TODO Auto-generated method stub
        //Database

            //Connessione al Db
            Connection c=new Database().Connect();

            //Prende la connessione
                try {
                   Statement st = c.createStatement();
                   String tot="insert into backupvalori values(null,'"+b.getRange_luminosita()+"','"+b.getRange_pressione()+"','"+
                   b.getRange_temperatura()+"','"+b.getRange_umidita()+"','"+b.getLuminosita_ottimale()+"','"+
                         b.getPressione_ottimale()+"','"+b.getTemperatura_ottimale()+"','"+b.getUmidita_ottimale()+"','"+""+"("+
                   'select ID from sensore order by ID desc limit 1"+")"+""+")";
                   st.executeUpdate(tot);
                } catch (SQLException e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
                }
}
```

3) Ripristino di un Sensore

RipristinoSensore.java

```
JButton btnRipristina = new JButton("Ripristina");
btnRipristina.setFont(new Font("Dialog", Font.ITALIC, 13));
btnRipristina.setBounds(167, 50, 89, 28);
contentPane.add(btnRipristina);
btnRipristina.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      IDvar = ID.getText();
      if (!IDvar.equals("")) {
         new GestoreSensoriController().ripristino(IDvar);
         dispose();
         JOptionPane.showMessageDialog(null,"Sensore ripristinato correttamente!");
      } else { JOptionPane.showMessageDialog(null,"Riempire tutti i campi");}
   }
});
```

*GestoreSensoriController.java

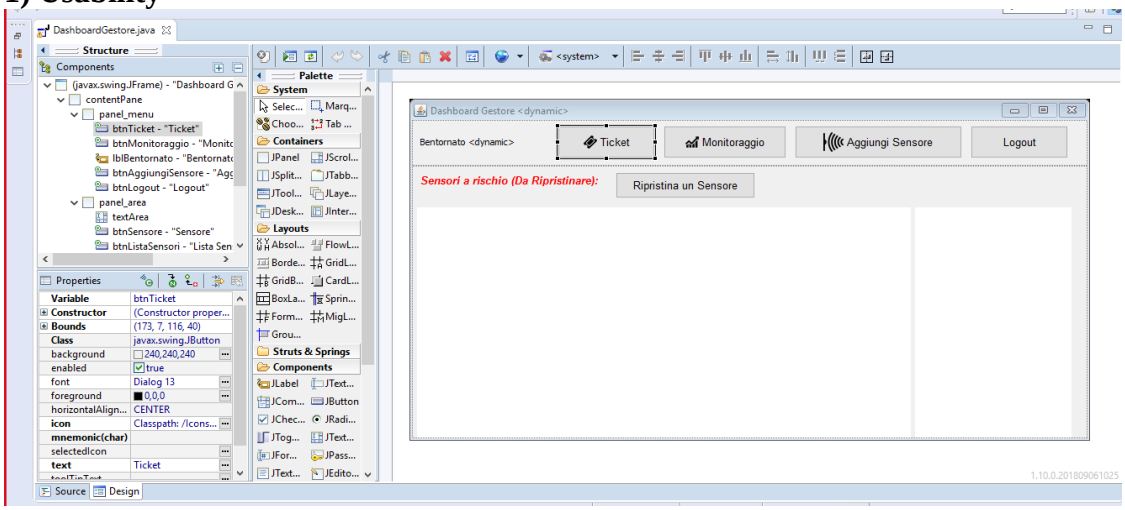
```
public void ripristino(String ID) {
    new GestoreDati().ripristino(ID);
}
```

*GestoreDati.java

```
280
281 public void ripristino(String ID) {
282     String[] a = new String[4];
283     Connection c=new Database().Connect();
284     Statement st;
285     Statement st2;
286     //Prende la connessione
287
288     try {
289         st = c.createStatement();
290         ResultSet rs = st.executeQuery("select LuminositàOttimale, PressioneOttimale, TemperaturaOttimale, "
291                 + "UmiditàOttimale from backupvalori where Sensore='"+ID+"'");
292         while(rs.next()) {
293             a[0]=rs.getString("TemperaturaOttimale");
294             a[1]=rs.getString("LuminositàOttimale");
295             a[2]=rs.getString("PressioneOttimale");
296             a[3]=rs.getString("UmiditàOttimale");
297         }
298     } catch (SQLException e) {
299         // TODO Auto-generated catch block
300         e.printStackTrace();
301     }
    try {
        st2 = c.createStatement();
        st2.executeUpdate("update sensore set Stato=0, Temperatura='"+a[0]+"', Luminosità='"+a[1]+"', "
                + "Pressione='"+a[2]+"', Umidità='"+a[3]+"' where ID='"+ID+"'");
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

NFR Coding:

1) Usability



2) Scalability

*GestoreUtenza.java

```
1  package Model.Components;
2
3  import java.sql.Connection;
14
15 public class GestoreUtenza implements GestoreUtenzaInterface{
16
17    @Override
18    public void setGestore(GestoreSensori g) {
19       //Database
20       //Connessione al Db
21       Connection c=new Database().Connect();
22       //Prende la connessione
23       try {
24          Statement st = c.createStatement();
25          String tot="insert into gestoresensori values('"+g.getUsername()+"','"+g.getPassword()+"','"+
26          g.getNome()+"','"+g.getCognome()+"','"+g.getTipo()+"','"+g.getLuogo_nascita()+"','"+
27                g.getData_nascita()+"','"+g.getEmail()+"','"+g.getTelefono()+"')";
28          st.executeUpdate(tot);
29       } catch (SQLException e) {
30          // TODO Auto-generated catch block
31          e.printStackTrace();
32       }
33    }
```

*GestoreUtenza.java

```
47    @Override
48    public void setAdmin(Amministratore a) {
49       //Database
50
51       //Connessione al Db
52       Connection c=new Database().Connect();
53
54       //Prende la connessione
55            try {
56               Statement st = c.createStatement();
57               String tot="insert into amministratore values('"+a.getUsername()+"','"+a.getPassword()+"','"+
58               a.getNome()+"','"+a.getCognome()+"','"+a.getData_nascita()+"','"+a.getLuogo_nascita()+"','"+
59                     a.getTelefono()+"','"+a.getEmail()+"','"+a.getChiave_recupero()+"')";
60               st.executeUpdate(tot);
61            } catch (SQLException e) {
62               // TODO Auto-generated catch block
63               e.printStackTrace();
64            }
65    }
```