UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica applicata

Corso di Laurea Magistrale in Ingegneria Informatica

DOCUMENTAZIONE PROGETTO

Progetto d'esame - Software Engineering

GRUPPO 14

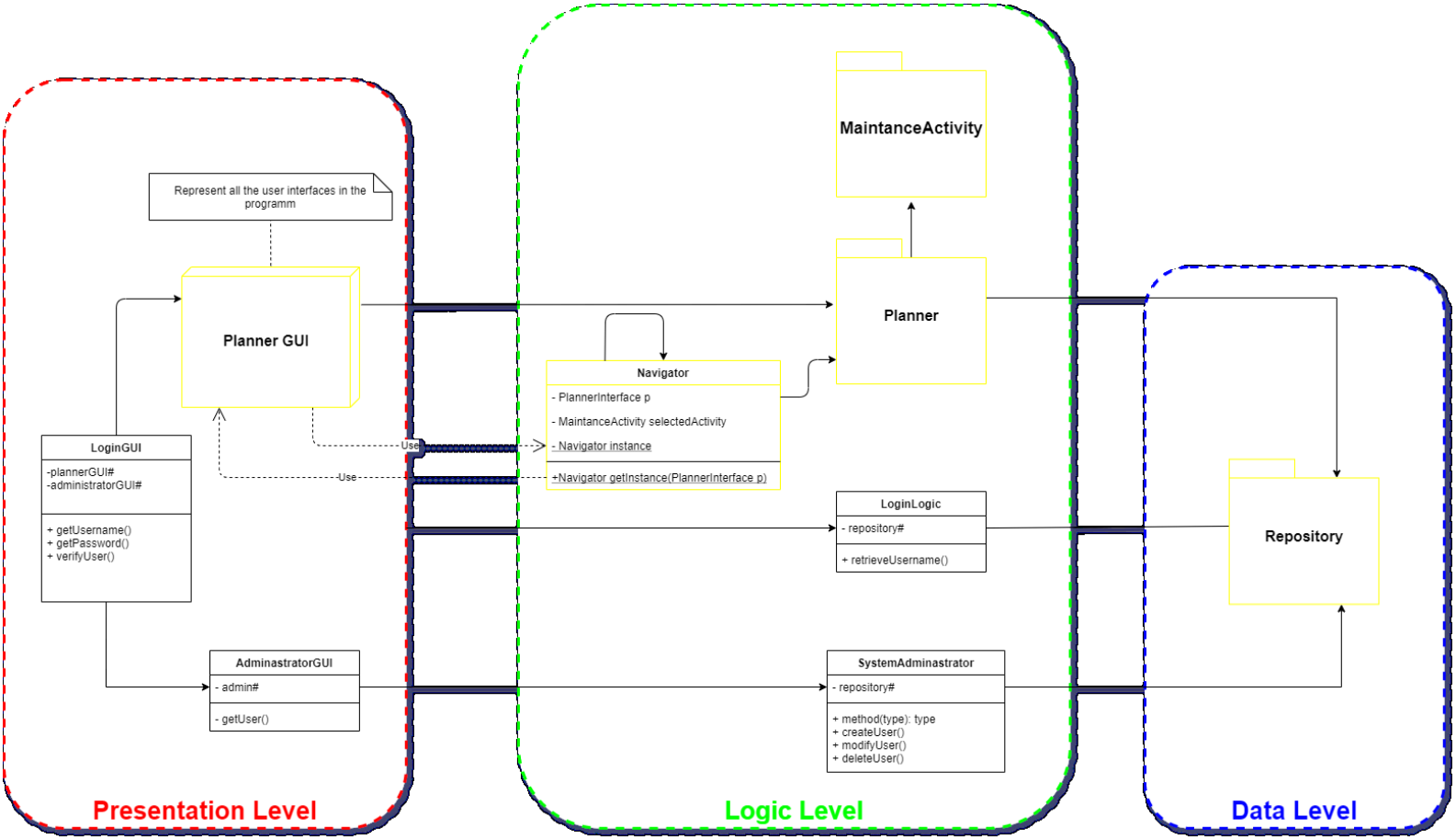
Professori:
Prof. Pasquale Foggia
Prof. Pierluigi Ritrovato

Membri del gruppo:
Enrico Frese - 0622701586
Gabriella De Marco - 0622701503
Catello Casillo - 0622701568
Nicolò Grieco - 0622701578

ANNO ACCADEMICO 2020/2021

SOMMARIO

1. Architettura ad alto livello

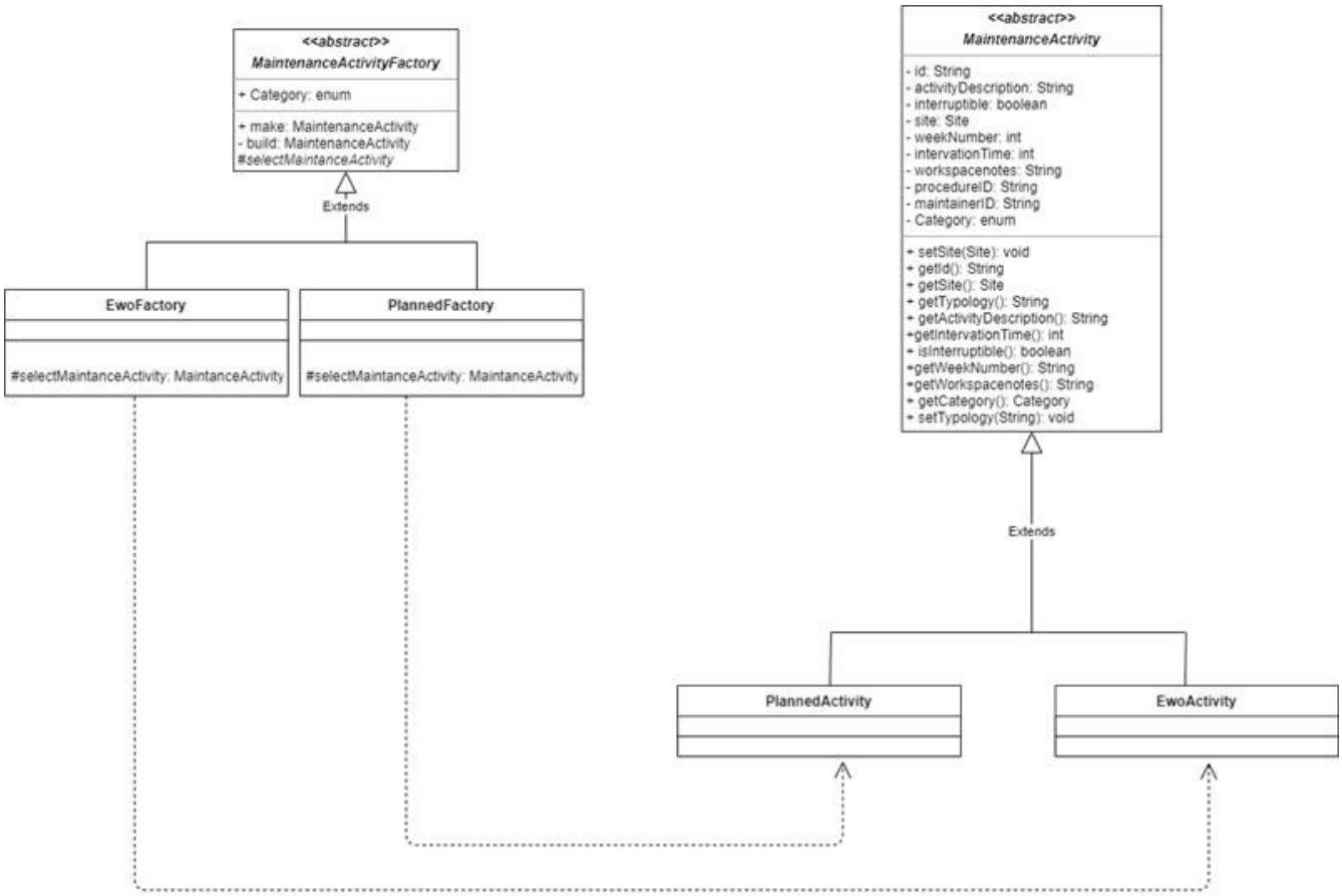


Il sistema si compone di tre livelli principali:

- Data Level: Rappresenta il livello dove vengono conservati i dati utilizzati dal sistema e contiene tutto il necessario per recuperarli e inserirne di nuovi.
- Logic Level: Rappresenta la logica di business dell'applicazione. In questo ultimo livello vengono effettuate tutte le trasformazioni necessarie alla visualizzazione dei dati provenienti dal livello precedente e salvataggio dei dati provenienti dal livello successivo.
- Presentation Level: Rappresenta il livello in cui vengono visualizzati i dati all'utente e tramite il quale può comunicare le sue esigenze al sistema.

Le parti del sistema sviluppate dal team di sviluppo fino a questo momento sono quelle evidenziate in giallo e verranno presentate in modo più approfondito nei punti seguenti del documento

2. Maintenance Activity



Per la creazione dell'oggetto Maintenance Activity è stato utilizzato il pattern Factory Method. In particolare, si possono individuare i seguenti componenti:

Maintenance Activity Factory: dichiara la Factory che avrà il compito di ritornare l'oggetto appropriato (la maintenance activity), tramite il metodo "make". Controlla il valore dell'enumerazione category se questo è "EWO" viene istanziato un oggetto EwoFactory, altrimenti se il valore è "PLANNED" viene istanziato un oggetto PlannedFactory. Su quest'ultimo verrà chiamato il metodo di build, il quale invoca il metodo astratto che è implementato nelle rispettive sottoclassi, per poi ritornare l'oggetto desiderato.

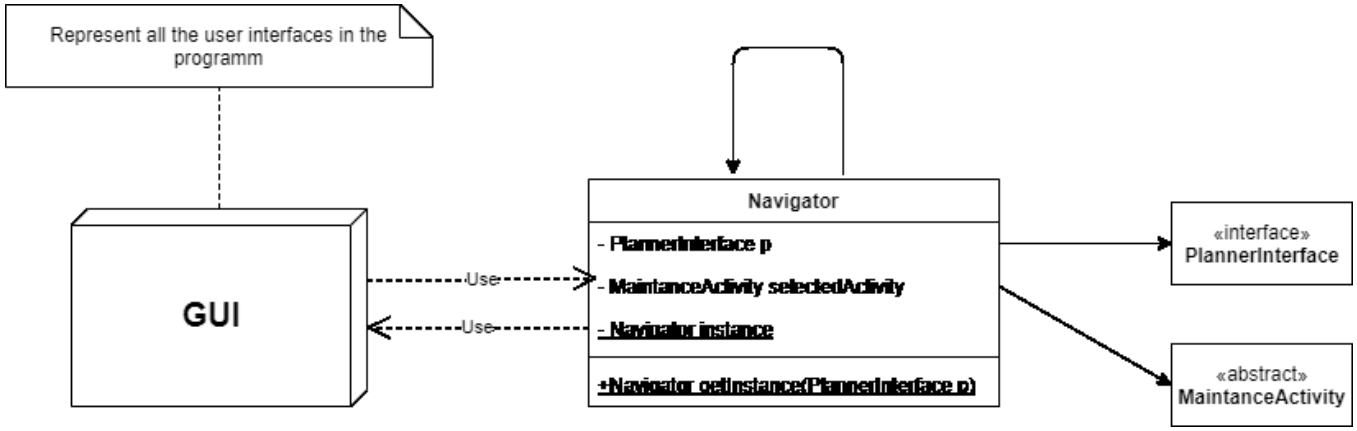
Ewo Factory: effettua l'override del metodo della MaintenanceActivity Factory al fine di ritornare l'istanza della classe concreta EwoActivity.

Planned Factory: effettua l'override del metodo della MaintenanceActivity Factory al fine di ritornare l'istanza della classe concreta PlannedActivity.

MaintenanceActivity: definisce gli attributi e i metodi degli oggetti creati dal metodo Factory.

La scelta di utilizzare il pattern Factory è stata dettata dall'intento di voler rendere il codice più flessibile e riusabile senza l'onere di dover istanziare specifiche classi, consentendo al client (nel nostro caso Planner), di non conoscere i dettagli della creazione degli oggetti.

3. Navigator



La classe Navigator rappresenta l'entità che permette ad ognuna delle interfacce utente presenti nel programma di visualizzarne un'altra.

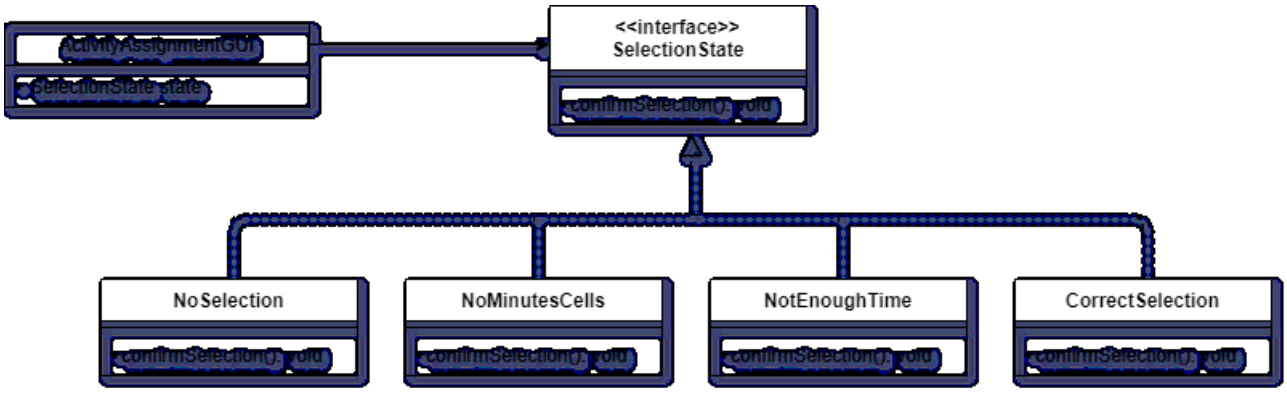
In particolare, il Navigator contiene un metodo per ognuna delle GUI definite che permette di istanziarla e visualizzarla e di eliminare dalla visualizzazione l'interfaccia chiamante. In questo modo le varie interfacce definite non dipendono dalle altre a cui sono collegate ma dal Navigator. Infatti, quest'ultimo nasconde all'interno dei suoi metodi di istanza la creazione delle interfacce tramite le chiamate ai costruttori veri e propri. Questo permette ad ogni interfaccia di passare all'interfaccia successiva meno parametri e soprattutto solo i parametri davvero necessari all'istanziazione della prossima interfaccia (e non anche parametri che sarebbero serviti alla creazione di interfacce non direttamente collegate a quella attuale).

Per realizzare il Navigator è stato seguito il design pattern Singleton in quanto tutte le interfacce ne devono condividere la stessa istanza visto che il Navigator mantiene al suo interno alcune scelte effettuate dall'utente in interfacce precedenti e necessarie per l'istanziazione della successive. Quindi il costruttore della Classe Navigator è privato e per ottenere una sua istanza è necessario chiamare il suo metodo statico *getIstance* che restituirà una nuovo Navigator se non ne è stato mai creato uno oppure verrà restituita l'istanza memorizzata nel campo statico *istance* .
Quando viene definita una nuova finestra per aggiungerla alla visualizzazione basterà definire un metodo del tipo *changeToNomeInterfaccia* che prende come parametri la finestra chiamante e i parametri necessari a costruire la nuova finestra che non siano già in qualche modo recuperabili dal Navigator. All'interno del metodo una volta istanziata l'interfaccia deve essere chiamato il metodo privato di utilità *changeInterface* che è l'effettivo responsabile dell'aggiunta alla visualizzazione della nuova interfaccia e della rimozione di quella corrente.

Dunque, una finestra per passare ad un'altra non dovrà far altro che ottenere un'istanza di Navigator con il metodo *getIstance* e chiamare su questa il metodo *changeToNomeInterfaccia* corrispondente alla successiva finestra da visualizzare (ovviamente passandogli tutti i parametri richiesti).

L'adozione di questo approccio ha portato ad una drastica riduzione delle dipendenze e a una maggiore semplicità nell'aggiunta, modifica e rimozione di nuove finestra alla visualizzazione.

4. ActivitityAssignementGUI e la gerarchia dei SelectionState



Questa parte di sistema si trova nel package ActvityAssignmentGUI facente parte del blocco di interfacce Planner GUI.
ActivityAssignmentGUI è l'interfaccia che permette la selezione da parte del Planner di come partizionare l'attività all'interno delle fasce orarie lavorative di un manutentore scelto precedentemente. Per confermare effettivamente la sua selezione e assegnazione, il Planner deve selezionare una o più celle della tabella corrispondenti alle fasce orarie e cliccare su bottone di SEND. Ovviamente a questo punto non tutte le selezioni sono valide quindi è stato deciso di utilizzare il pattern State per rappresentare i diversi stati della selezione. In particolare:

- NoSelection: nessuna selezione. Rappresenta lo stato inziale oppure lo stato che segue un'operazione di clear della selezione sulla tabella. Fornisce un'implementazione vuota del metodo *confirmSelection()* (visto che non è una selezione né valida né invalida o almeno non ancora).
- NoMinutesCells: il Planner ha selezionato delle celle che non rappresentano fasce orarie (come il nome del manutentore o la sua skillCompliance). Implementa il metodo *confirmSelection()* mostrando una finestra con un messaggio per portare all'attenzione del Planner l'errore commesso sulla sua selezione.
- NotEnoughTime: il Planner ha selezionato delle celle che corrispondono a fasce orarie la cui somma del tempo rimanente non è sufficiente ad eseguire tutta l'attività scelta

precedentemente. Implementa il metodo *confirmSelection()* mostrando una finestra con un messaggio per portare all'attenzione del Planner l'errore commesso sulla sua selezione.

- CorrectSelecion: non rappresenta uno stato di errore ed è una scelta valida. Implementa il metodo *confirmSelection()* effettivamente tentando di effettuare le seguenti operazioni l'assegnazione dell'attività allo specifico manutentore e delle frazioni delle attività alla fasce orarie. Se queste operazioni falliscono allora il Planner verrà informato trami un'apposita finestra. Mentre se hanno successo, verrà visualizzata un'altra finestra per rincuorare il Planner sull'esito positivo delle operazioni stesse e alla sua conferma verrà riportato all'interfaccia SelectActvityGUI.

È stato scelto di utilizzare questo pattern architetturale per aumentare la manutenibilità del codice in modo che se in futuro qualcuno volesse cambiare il tipo di selezione aggiungere nuovi stati o rimuoverne di vecchi sia molto più semplice.

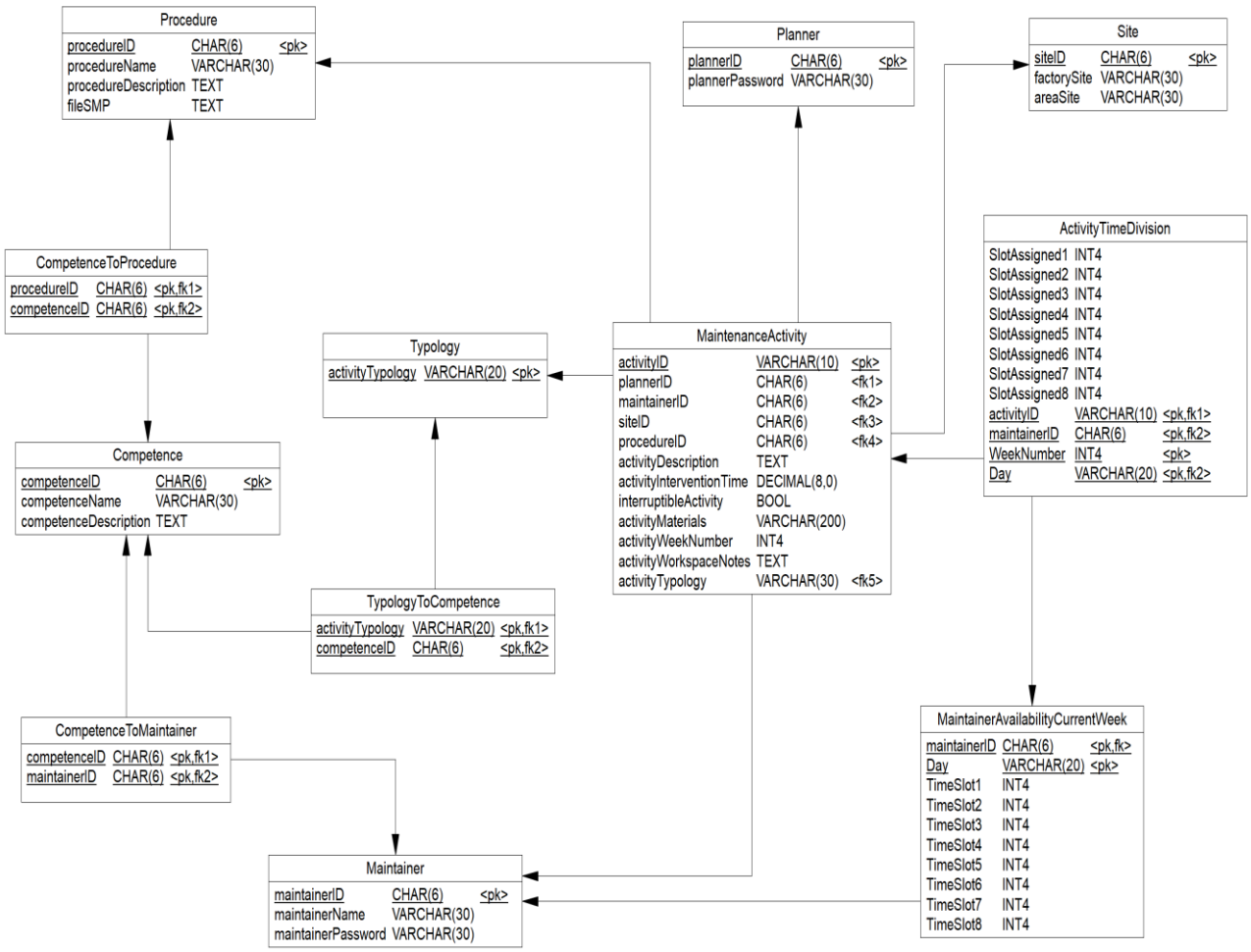
5. Planner

Questo package contiene tutte le classi necessarie ad effettuare le trasformazioni sui dati gestiti da un pianificatore. In particolare, sono presenti operazioni per la trasformazione dei dati recuperati dal repository per visualizzare le attività di manutenzione in modo agevole, visualizzare la disponibilità settimanale dei manutentori ed altro ancora. Sono presenti anche le trasformazioni per il flusso inverso dei dati ovvero quelle che permettono di salvare i dati e le scelte fatte dal pianificatore tramite le interfacce all'interno del repository.

Di questo parte del sistema purtroppo siamo riusciti a completare solo una parte del refactory che avevamo pianificato di portare a termine. Infatti, abbiamo scomposto per ora la classe originale in un'interfaccia PlannerInterface, un classe astratta PlannerAbstract e una classe concreta. L'idea era quella di utilizzare l'interfaccia per utilizzare diversi tipi di Planner (a seconda della tipologia dei dati che vengono gestiti) attraverso il meccanismo del polimorfismo arrivando a una struttura simili alle classi del repository.

Nonostante l'operazione di refactoring non sia completa in questo modo siamo comunque riusciti a spezzare la catena di dipendenze che si diramava verso le classi che rappresentano le attività di manutenzione e verso quelle che rappresentano il repository.

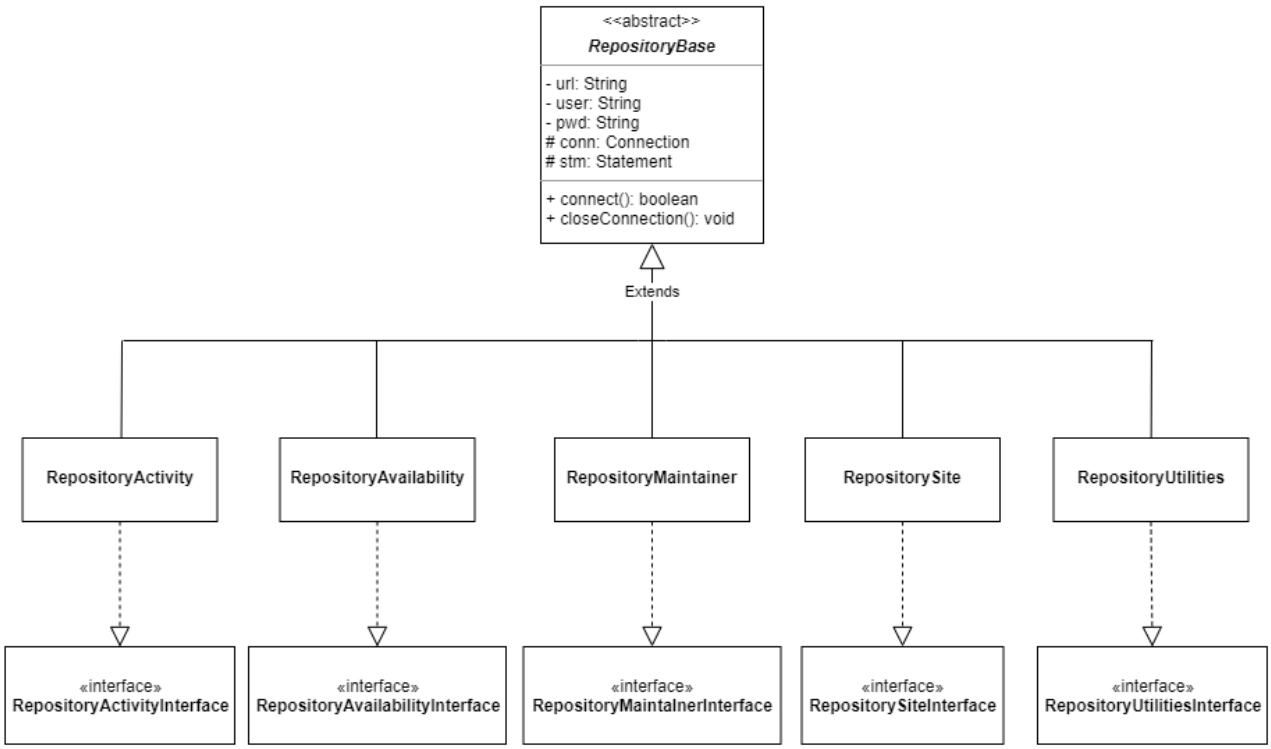
6. Schema logico - PostgreSQL



Per la realizzazione del progetto è stato scelto di utilizzare un DBMS relazionale: PostgreSQL. Viene presentato nell'immagine lo schema logico adottato per la realizzazione del Database. Come vincoli aggiuntivi sono stati inseriti dei check su vari parametri:

- il numero della settimana (weekNumber) deve essere compreso tra 1 e 52 entrambi inclusi;
- la disponibilità nell'ora di lavoro della giornata di un manutentore (indicata con TimeSlot) deve essere compresa tra 0 e 60 (questi indicano i minuti rimaneti);
- la quantità di tempo che viene assegnata ad una specifica attività di manutenzione in una determinata ora (indicata con SlotAssigned) deve essere compresa tra 0 e 60 (questi indicato i minuti assegnati in quell'ora);
- i giorni della settimana inseriti sono questi presentati nel seguente elenco: "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday".

7. Repository Package



Nel package *Repository* sono presenti le varie classi attraverso le quali il livello logico del sistema scambia informazioni con il Database collegato. L'immagine sovrastante descrive la struttura gerarchica utilizzata. Vi è una classe astratta *RepositoryBase* che permette l'apertura e la chiusura della connessione con il DB, questa viene estesa da varie sottoclassi, le quali sono state così divise e strutturate in modo da raggruppare le operazioni comuni e significative da effettuare sul DB. Più nello specifico *RepositoryActivity* si occupa di prendere le varie attività di manutenzione presenti sul DB, di inserirne di nuove, di eliminarle e di modificarle, inoltre permette di ottenere il valore di un singolo parametro dell'attività di manutenzione. *RepositoryAvailability,* invece, permette varie operazioni per gestire la disponibilità di un manutentore indicandone o il giorno o il numero di settimana, inoltre qui viene gestita l'assegnazione di un'attività di manutenzione, la sua cancellazione e le operazioni per distribuire quanto tempo di un'attività viene assegnato ad una determinata ora di lavoro di un manutentore. A seguire, *RepositoryMaintainer* si occupa dell'inserimento di nuovi manutentori e della loro disponibilità, recupera i manutentori già inseriti sul DB e permette di ottenere le competenze di un manutentore e i suoi parametri. *RepositorySite* permette di recuperare i Siti già presenti sul DB e di ottenere i loro singoli parametri (id,area,factory). Infine, *RepositoryUtilities* mette a disposizione delle operazioni di supporto tra cui recuperare le informazioni dal DB sulle tipologie e competenze presenti, è inoltre possibile recuperare i loro singoli parametri e sapere quali competenze sono necessarie per una determinata tipologia.

Tutte le sotto classi di *RepositoryBase* sopracitate implementano un'interfaccia specifica che ne definisce i metodi da utilizzare in modo da eliminare le dipendenze.

Infine nel package è presente un main denominato *DemoAdminInsertNewMaintainer* che simula l'inserimento di vari manutentori all'interno del sistema.