



UNIVERSITÀ DEGLI STUDI DI  
BERGAMO  
SCUOLA DI INGEGNERIA

**Gestione turnazione dipendenti**

Informatica IIIB  
A.A. 2017-2018

**Documentazione progetto**

Studenti:

**Arif Abdelmajid**  
Mat.1030217

**Bombarda Andrea**  
Mat.1029969

**Paravisi Federica**  
Mat.1029712



# Indice

|          |   |           |
|----------|---|-----------|
| <b>I</b> | <b>Fase 0 - Fase 1</b>  | <b>7</b>  |
| <b>1</b> | <b>Requisiti</b>  | <b>9</b>  |
| 1        | Introduzione . . . . .  | 9         |
| 2        | Esempio di file .XML . . . . .                                      | 10        |
| 3        | Tool chain . . . . .  | 11        |
| <b>2</b> | <b>Specifiche</b>   | <b>13</b> |
| 1        | Introduzione . . . . .  | 13        |
| 2        | Funzionalità richieste . . . . .                                    | 13        |
| 2.1      | Gestione dipendenti . . . . .                                       | 13        |
| 2.2      | Gestione dati base dell'azienda . . . . .                           | 14        |
| 2.3      | Gestione turni dipendenti . . . . .                                 | 14        |
| 2.4      | Gestione file XML . . . . .   | 15        |
| 2.5      | Gestione orari . . . . .  | 15        |
| 3        | Analisi casi d'uso . . . . .  | 16        |
| 3.1      | UC1: Inserimento dipendente . . . . .                               | 16        |
| 3.2      | UC2: Cancellazione dipendente . . . . .                             | 18        |
| 3.3      | UC3: Modifica dipendente . . . . .                                  | 18        |
| 3.4      | UC4: Inserimento nuova attività . . . . .                           | 20        |
| 3.5      | UC5: Cancellazione di una attività . . . . .                        | 21        |
| 3.6      | UC6: Modifica di una attività . . . . .                             | 21        |
| 3.7      | UC7: Allocazione dipendenti per ogni attività . . . . .             | 22        |
| 3.8      | UC8: Generazione turnazione settimanale . . . . .                   | 23        |
| 3.9      | UC9: Visualizzazione turnazione settimanale . . . . .               | 24        |
| 3.10     | UC10: Visualizzazione turnazione giornaliera . . . . .              | 26        |
| 3.11     | UC11: Cambio turno dipendente . . . . .                             | 27        |
| 3.12     | UC12: Gestione dei dati provenienti dalla timbratrice . . . . .     | 28        |
| 3.13     | UC13: Gestione straordinari . . . . .                               | 30        |
| 3.14     | UC14: Gestione giorni di malattia/ferie . . . . .                   | 32        |
| 3.15     | UC15: Visualizzazione e calcolo giornaliero ore di lavoro . . . . . | 34        |
| 3.16     | UC16: Visualizzazione e calcolo mensile ore di lavoro . . . . .     | 36        |
| 3.17     | UC17: Applicazione incremento percentuale straordinari . . . . .    | 38        |
| 3.18     | UC18: Calcolo stipendio . . . . .                                   | 38        |
| 3.19     | UC19: Generazione report dipendente . . . . .                       | 39        |
| 3.20     | UC20: Prenotazione per straordinari . . . . .                       | 40        |
| 3.21     | UC21: Verifica copertura turni assegnati . . . . .                  | 41        |
| 4        | Use case diagram . . . . .  | 43        |

|            |   |           |
|------------|---|-----------|
| <b>3</b>   | <b>Architettura</b>                                       | <b>45</b> |
| 1          | Deployment diagram . . . . .                              | 45        |
| 1.1        | Architettura hardware . . . . .                           | 45        |
| 1.2        | Architettura software . . . . .                           | 46        |
| 1.3        | Analisi dei componenti . . . . .                          | 48        |
| <b>II</b>  | <b>Fase 2</b>   | <b>59</b> |
| <b>4</b>   | <b>Introduzione</b>                                       | <b>61</b> |
| 1          | Scelta della funzione da implementare . . . . .           | 61        |
| 2          | Passi principali della funzionalità . . . . .             | 61        |
| <b>5</b>   | <b>Creazione del Database</b>                             | <b>63</b> |
| 1          | Creazione del DB . . . . .                                | 63        |
| <b>6</b>   | <b>Parser XML ed inserimento dati grezzi in DB</b>        | <b>67</b> |
| 1          | Formato file XML . . . . .                                | 67        |
| 2          | Set-up preventivo . . . . .                               | 68        |
| 3          | Implementazione del parser . . . . .                      | 68        |
| 4          | Funzionamento della classe XMLHandler . . . . .           | 69        |
| 4.1        | Identificazione dei tag . . . . .                         | 70        |
| 4.2        | Selezione del contenuto del tag . . . . .                 | 71        |
| 4.3        | Chiusura del tag . . . . .                                | 71        |
| 4.4        | Risultato del Parsing . . . . .                           | 72        |
| 5          | Inserimento dei dati grezzi nel database . . . . .        | 72        |
| <b>7</b>   | <b>Aggregazione degli orari</b>                           | <b>73</b> |
| 1          | Introduzione . . . . .                                    | 73        |
| 2          | Funzione di aggregazione . . . . .                        | 73        |
| 2.1        | Funzione di creazione delle righe vuote . . . . .         | 74        |
| 2.2        | Funzione di posizionamento degli orari . . . . .          | 75        |
| 2.3        | Funzione di elaborazione degli orari . . . . .            | 77        |
| 2.4        | Funzione di eliminazione dei dati grezzi . . . . .        | 79        |
| 2.5        | Analisi della complessità generale . . . . .              | 79        |
| <b>8</b>   | <b>Test ed analisi del componente implementato</b>        | <b>81</b> |
| 1          | Analisi statica . . . . .                                 | 81        |
| 2          | Analisi dinamica - testing . . . . .                      | 81        |
| 2.1        | Test della classe DateValidator . . . . .                 | 82        |
| 2.2        | Test della classe TimeValidator . . . . .                 | 84        |
| 2.3        | Test della funzionalità di parsing del file XML . . . . . | 86        |
| 2.4        | Test della classe RecordAggregator . . . . .              | 88        |
| <b>III</b> | <b>Fase 3</b>   | <b>93</b> |
| <b>9</b>   | <b>Introduzione</b>                                       | <b>95</b> |
| 1          | Scelta della funzione da implementare . . . . .           | 95        |
| 2          | Aggiornamento dei dati nel database . . . . .             | 95        |

|  |            |
|--|------------|
| <b>10 Progettazione algoritmo</b>                      | <b>99</b>  |
| 1 Modello del problema . . . . .                       | 99         |
| 1.1 Indici utilizzati . . . . .                        | 99         |
| 1.2 Variabili decisionali . . . . .                    | 99         |
| 1.3 Variabili considerate . . . . .                    | 100        |
| 1.4 Vincoli . . . . .                                  | 100        |
| 1.5 Funzione obiettivo . . . . .                       | 100        |
| 2 Algoritmo greedy per la risoluzione . . . . .        | 100        |
| 2.1 Pseudocodice dell'algoritmo . . . . .              | 101        |
| 2.2 Analisi della complessità dell'algoritmo . . . . . | 102        |
| 2.3 Flow-chart dell'algoritmo . . . . .                | 103        |
| 2.4 Risultato atteso . . . . .                         | 104        |
| 3 Implementazione dell'algoritmo . . . . .             | 105        |
| 3.1 Interfaccia Scheduler . . . . .                    | 105        |
| 3.2 Classe StaffScheduler . . . . .                    | 106        |
| 3.3 Classe ListIntersect<T> . . . . .                  | 108        |
| 4 Risultato ottenuto . . . . .                         | 108        |
| <b>11 Test ed analisi del componente implementato</b>  | <b>111</b> |
| 1 Analisi statica . . . . .                            | 111        |
| 2 Analisi dinamica - testing . . . . .                 | 112        |
| 2.1 Test della classe ListIntersect . . . . .          | 112        |
| 2.2 Test della classe StaffScheduler . . . . .         | 114        |



## Parte I

### Fase 0 - Fase 1





# Capitolo 1

## Requisiti

### 1 Introduzione

Il cliente richiede lo sviluppo di un'applicazione per la gestione automatizzata degli orari e dei turni dei propri dipendenti.

Il processo produttivo dell'azienda committente è attivo 7 giorni a settimana. Ciascuna giornata è suddivisa in tre turnazioni:

- 00:00 - 08:00
- 08:00 - 16:00
- 16:00 - 24:00

Nel corso del proprio turno lavorativo, ogni dipendente ha diritto a 30 minuti continuativi di pausa.

Al fine di organizzare al meglio le lavorazioni, per ogni attività del processo produttivo, l'azienda definisce settimanalmente il numero di dipendenti necessari contemporaneamente per il suo svolgimento.

L'applicazione realizzata dovrà permettere al committente di svolgere le seguenti macro operazioni:

1. Gestione dei dati dei dipendenti.
2. Calcolo degli orari mensili e degli stipendi dei dipendenti.
3. Allocazione dei dipendenti sui turni.

Dei dipendenti si vogliono gestire i principali dati anagrafici (nome, cognome, data di nascita, etc.), un elenco di attività eseguibili dal dipendente, le informazioni riguardanti la paga base oraria ed il numero di ore settimanali contrattuali.

Nella sede del cliente è presente una timbratrice che, una volta al giorno, memorizza all'interno di un percorso di rete noto un file `.XML` (un esempio del quale è riportato nella sezione 2 del capitolo corrente) contenente tutte le timbrature effettuate dai dipendenti durante le 24 *h* precedenti (indicando anche se è una

timbratura di tipo "IN" o "OUT"). Ogni dipendente ha, infatti, un badge personale che permette la timbratura ad ogni entrata o uscita dall'area produttiva (compresa la pausa intermedia).

Le informazioni raccolte dalla timbratrice tramite file `.XML` dovranno essere utilizzate dall'applicazione per il calcolo degli orari mensili di ciascun dipendente. In particolare è richiesto il calcolo di:

- Ore di lavoro ordinario
- Ore di lavoro straordinario
- Ore di malattia
- Ore di ferie/permessi

Queste informazioni devono essere utilizzate anche per il calcolo della paga mensile, considerando che le ore di straordinario e le ore notturne (prima delle 08:00 e dopo le 22:00) sono pagate con un incremento percentuale, configurabile dall'utente.

Il committente richiede inoltre di avere una tolleranza di 5 minuti sugli orari (ad esempio un dipendente che deve iniziare alle 08:00, potrà timbrare fino alle 08:05 senza che venga conteggiato alcun permesso).

Attualmente, il cliente calcola gli orari dei dipendenti in modo manuale, convertendo il file `.XML` in `.CSV` che viene successivamente importato in Microsoft Excel®.

Il processo di calcolo degli orari mensili dovrà essere eseguito in automatico dall'applicazione, ma dovrà essere anche presente una sezione in cui l'addetto dell'ufficio amministrazione potrà eseguire una revisione dei calcoli, o una modifica manuale dei dati.

Inoltre l'applicazione dovrà consentire la stampa di un report PDF contenente il quadro generale degli orari mensili di ciascun dipendente.

Per quanto riguarda l'allocazione dei dipendenti sui turni, attualmente, il committente gestisce le turnazioni tramite accordo privato tra i dipendenti, senza alcuna automatizzazione del processo. L'applicazione dovrà, considerando il numero di dipendenti necessari in ogni turno e per ogni attività, fornire un piano settimanale, avendo cura di non superare il numero di ore contrattuali previste per ciascun dipendente.

Dovrà essere anche prevista la possibilità di fare un cambio turno: quando un dipendente non può presentarsi al turno che gli è stato assegnato, deve essere ricalcolato il piano settimanale, in modo da coprire lo slot lasciato scoperto.

Infine, avendo a disposizione i dati delle timbrature dei dipendenti e la programmazione dei turni, è richiesta l'implementazione di un sistema di controllo, utilizzabile al termine del mese, per verificare l'effettiva corrispondenza tra turni assegnati a ciascun dipendente e relativa presenza in azienda.

## 2 Esempio di file `.XML`

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <records date="30/04/2018">
3   <record type = "IN">
4     <badgeID>00002</badgeID>
5     <time>00:00:00</time>
6   </record>
7   <record type = "IN">
8     <badgeID>00004</badgeID>
9     <time>07:48:00</time>
10  </record>
11  <record type = "OUT">
12    <badgeID>00002</badgeID>
13    <time>08:01:00</time>
14  </record>
15  <record type = "IN">
16    <badgeID>00001</badgeID>
17    <time>08:02:00</time>
18  </record>
19  <record type = "OUT">
20    <badgeID>00005</badgeID>
21    <time>10:30:00</time>
22  </record>
23 </records>
```

Codice 1.1: Esempio di file .XML generato dalla timbratrice

### 3 Tool chain

Per la realizzazione del presente software verranno utilizzati i seguenti strumenti:

- Modellazione
  1. **Use-case diagram, deployment diagram, component diagram:** *UMLet* (<http://www.umlet.com>), strumento Java open source, molto intuitivo ed utile per la realizzazione di diagrammi (principalmente UML).
  2. **Class diagram, sequence diagram:** *ObjectAid UML Explorer*, plugin di Eclipse in grado di generare in automatico i diagrammi.
  3. **Database diagram:** *Vertabelo* (<http://www.vertabelo.com>), strumento online utile per la modellazione dei diagrammi di database e per la generazione automatica dei comandi DDL in codice SQL.
- Implementazione software
  1. **DBMS:** *MySQL*, distribuito insieme all'applicativo Apache XAMPP (<https://www.apachefriends.org>)
  2. **Linguaggio di programmazione e IDE:** *JAVA*, tramite l'IDE *Eclipse* (<http://www.eclipse.org>).
  3. **Interfaccia grafica:** *Windowbuilder*, plugin Eclipse che permette la progettazione di interfacce grafiche in maniera drag and drop (<https://eclipse.org/windowbuilder/download.php>).
  4. **Persistenza:** realizzata tramite *JPA 2.0*.

5. **Parsing XML:** *SAXParser*, già presente nella libreria standard di Java (<https://docs.oracle.com/javase/7/docs/api/javax/xml/parsers>).
  6. **Classi e interfacce JCF:** *List*, *ArrayList* e *Vector*, contenute nel package `java.util` (<http://docs.oracle.com/javase/tutorial/collections>). Sono inoltre stati utilizzati alcune funzioni, contenute nella libreria, per lavorare sulle `Collections`, come la `Collections.sort`.
- Analisi del software
    1. **Analisi statica:** *Stan4J*, plugin di Eclipse (<http://stan4j.com>).
    2. **Analisi dinamica:** *JUnit*, plugin integrato di default in Eclipse (<http://stackoverflow.com/questions/1962567/junit-eclipse-plugin>).
  - Documentazione e varie
    1. **Versioning:** *GitHub*, gestito grazie alla relativa applicazione desktop (<https://github.com>).
    2. **Documentazione:** *LaTeX*, con scrittura tramite l'editor *Texpad* (<https://www.texpad.com>)

# Capitolo 2

## Specifiche

### 1 Introduzione

**Importanza delle funzioni** Di seguito vengono riportate le varie funzioni richieste per soddisfare i requisiti posti dal committente. Non tutte le funzioni, però, sono importanti allo stesso livello, ma alcune richiedono di essere implementate prima di altre.

In particolare verrà definito un livello di importanza in base alla seguente classificazione:

- Funzioni con alta priorità: sono le funzioni sulle quali si basa il funzionamento "base" del programma che, quindi, richiedono di essere realizzate prima di quelle con media priorità.
- Funzioni con media priorità: sono le funzioni che richiedono di essere realizzate prima di quelle con bassa priorità. Date le caratteristiche di queste funzioni è possibile implementarle parallelamente alle altre funzioni.
- Funzioni con bassa priorità: sono le funzioni "di contorno", dalle quali non dipende alcuna altra funzione.

### 2 Funzionalità richieste

#### 2.1 Gestione dipendenti

| Codice | Importanza | Nome funzione             |
|--------|------------|---------------------------|
| DD01   | Alta       | Inserimento dipendente    |
| DD02   | Bassa      | Cancellazione dipendente  |
| DD03   | Media      | Modifica dipendente       |
| DD04   | Bassa      | Invio mail riepilogo dati |
| DD05   | Bassa      | Visualizzazione dati      |
| DD06   | Bassa      | Stampa dati               |

|      |       |                               |
|------|-------|-------------------------------|
| DD07 | Bassa | Statistiche dipendenti        |
| DD08 | Alta  | Inserimento dati nel database |

Tabella 2.1: Funzionalità per la gestione dei dipendenti

## 2.2 Gestione dati base dell'azienda

| Codice | Importanza | Nome funzione                           |
|--------|------------|---|
| DA01   | Alta       | Inserimento attività                    |
| DA02   | Alta       | Cancellazione attività                  |
| DA03   | Media      | Modifica attività                       |
| DA04   | Alta       | Modifica numero dipendenti per attività |
| DA05   | Bassa      | Statistiche azienda                     |
| DA06   | Bassa      | Visualizzazione dati azienda            |
| DA07   | Bassa      | Stampa dati azienda                     |
| DA08   | Alta       | Inserimento dati nel database           |

Tabella 2.2: Funzionalità per la gestione dei dati base dell'azienda

## 2.3 Gestione turni dipendenti

| Codice | Importanza | Nome funzione                          |
|--------|------------|--|
| TD01   | Bassa      | Visualizzazione turnazione settimanale |
| TD02   | Media      | Visualizzazione turnazione giornaliera |
| TD03   | Bassa      | Stampa report turnazione settimanale   |
| TD04   | Bassa      | Stampa report turnazione giornaliera   |
| TD05   | Bassa      | Invio mail turnazione dipendente       |
| TD06   | Alta       | Generazione turnazione settimanale     |
| TD07   | Alta       | Generazione turnazione giornaliera     |
| TD08   | Alta       | Assegnazione dipendente ad un turno    |
| TD09   | Alta       | Richiesta cambio turno dipendente      |
| TD10   | Alta       | Ricalcolo turnazione dopo cambio turno |
| TD11   | Bassa      | Invia mail cambio turno                |
| TD12   | Media      | Allarme turnazione non possibile       |

|      |       |   |
|------|-------|---|
| TD13 | Bassa | Visualizzazione turnazione settimana dipendente |
| TD14 | Media | Visualizzazione turnazione giorno dipendente    |
| TD15 | Media | Salvataggio dati turnazioni in database         |

Tabella 2.3: Funzionalità per la gestione dei turni dei dipendenti

## 2.4 Gestione file XML

| Codice | Importanza | Nome funzione                          |
|--------|------------|--|
| FM00   | Alta       | Parsing file XML                       |
| FM01   | Media      | Estrazione e selezione campi           |
| FM02   | Alta       | Memorizzazione timbrature nel database |
| FM03   | Bassa      | Modifica timbrature dal database       |
| FM04   | Bassa      | Eliminazione timbrature dal database   |
| FM05   | Media      | Visualizzazione dei dati grezzi        |

Tabella 2.4: Funzionalità per la gestione dei file XML

## 2.5 Gestione orari

| Codice | Importanza | Nome funzione                                |
|--------|------------|--|
| SM00   | Alta       | Calcolo giornaliero ore lavoro ordinario     |
| SM01   | Alta       | Calcolo giornaliero ore lavoro straordinario |
| SM02   | Alta       | Calcolo giornaliero ore ferie/permessi       |
| SM03   | Bassa      | Revisione e controllo generale               |
| SM04   | Alta       | Calcolo giornaliero ore malattia             |
| SM05   | Media      | Calcolo mensile ore lavoro ordinario         |
| SM06   | Media      | Calcolo mensile ore lavoro straordinario     |
| SM07   | Media      | Calcolo mensile ore ferie/permessi           |
| SM08   | Media      | Calcolo mensile ore malattia                 |
| SM09   | Alta       | Inserimento dati giornalieri nel database    |
| SM10   | Alta       | Inserimento dati mensili nel database        |
| SM11   | Media      | Applicazione incremento percentuale          |
| SM12   | Bassa      | Calcolo stipendio mensile                    |

|      |       |   |
|------|-------|---|
| SM13 | Bassa | Generazione report mensile dipendente       |
| SM14 | Alta  | Inserimento malattia/ferie                  |
| SM15 | Alta  | Aggiornamento malattia/ferie                |
| SM16 | Alta  | Eliminazione malattia/ferie                 |
| SM17 | Bassa | Inserimento ore straordinarie necessarie    |
| SM18 | Bassa | Aggiornamento ore straordinarie necessarie  |
| SM19 | Bassa | Eliminazione ore straordinarie necessarie   |
| SM20 | Alta  | Modifica dati giornalieri nel database      |
| SM21 | Alta  | Eliminazione dati giornalieri nel database  |
| SM22 | Media | Modifica dati mensili nel database          |
| SM23 | Media | Eliminazione dati mensili nel database      |
| SM24 | Media | Visualizzazione dati mensili dipendente     |
| SM25 | Bassa | Verifica copertura turno assegnato          |
| SM26 | Media | Visualizzazione dati straordinari           |
| SM27 | Media | Visualizzazione dati giornalieri dipendente |
| SM28 | Bassa | Prenotazione per ore di straordinario       |

Tabella 2.5: Funzionalità per la gestione degli orari dei dipendenti

### 3 Analisi casi d'uso

#### 3.1 UC1: Inserimento dipendente

**Descrizione** Si vuole inserire un nuovo dipendente.

**Requisiti coperti** DD01, DD03-DD08, DA01, DA08

**Attori coinvolti** Gestore del personale

**Precondizioni** Il dipendente non è presente nel database.

**Postcondizioni** Il dipendente è stato aggiunto al database.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale preme il tasto “Aggiungi dipendente”.
2. Il gestore del personale compila la form in cui sono richiesti i dati del dipendente (nome, cognome, data di nascita, data di inizio e fine contratto,



indirizzo email, numero di ore contrattuali a settimana, attività svolte, numero del badge, etc.).

3. Il gestore del personale preme “OK” per confermare l’inserimento del dipendente.

(a) Il sistema mostra il riepilogo dei dati inseriti per il dipendente.

4. Il gestore del personale preme “Conferma” per confermare l’inserimento del dipendente.

(a) I dati del dipendente vengono inseriti all’interno del database.

(b) Il dipendente riceve una e-mail contenente la conferma di inserimento ed il riepilogo delle informazioni inserite.

(c) Il sistema mostra un riepilogo indicante, per ogni attività disponibile, quali e quanti dipendenti sono disponibili.

### Alternative

- Inserimento dati errati

- Al passo (2) sono inseriti dati errati, in questo caso l’inserimento è annullato e viene lanciato un messaggio di errore.
- In alternativa ci si può accorgere di aver inserito dati errati al passo (3). In questo caso il gestore del personale può cliccare su “Modifica dati”, tornando, quindi, al passo (2).
- Con “dati errati” si intende nome, cognome, posizione vuoti o formato delle date diverso da gg/mm/aaaa.

- Dipendente a tempo indeterminato

- Al passo (2) non viene compilato il campo “data fine contratto”.
- Quando si clicca “Prosegui” viene lanciato un messaggio di avviso per informare del campo vuoto, ma è comunque possibile proseguire.

- Attività non presente

- Al passo (2) il gestore del personale non trova l’attività da assegnare al dipendente.
- Il gestore del personale clicca sul pulsante “Nuova Attività”.
  - \* Da qua si procede come da UC4.

- Dipendente senza indirizzo e-mail

- Al passo (2) non viene compilato il campo “indirizzo e-mail”.
- Quando si clicca “Prosegui” viene lanciato un messaggio di avviso per informare del campo vuoto, ma è comunque possibile proseguire.

**Estensioni**

- Stampa dati dipendente
  - Al passo (3) il gestore del personale può cliccare sul pulsante “Stampa”, per stampare la scheda di riepilogo contenente i dati inseriti per il nuovo dipendente.

**3.2 UC2: Cancellazione dipendente**

**Descrizione** Si vuole eliminare dal database un dipendente.

**Requisiti coperti** DD02, DD05

**Attori coinvolti** Gestore del personale

**Precondizioni** Vari dipendenti sono presenti nel database.

**Postcondizioni** Il dipendente viene eliminato dal database.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale clicca su “Lista dipendenti” nella pagina di gestione dei dipendenti.
  - (a) Viene visualizzata la lista di tutti i dipendenti.
2. Il gestore del personale seleziona il dipendente che vuole eliminare.
3. Il gestore del personale clicca su “Elimina dipendente”.
  - (a) E’ richiesta la conferma dell’azione: se l’azione è confermata, il dipendente viene eliminato dal database.
  - (b) Viene aggiornata la lista di tutti i dipendenti, in modo che si possa eventualmente procedere ad una ulteriore eliminazione.

**Alternative**

- Dipendente con turnazione assegnata
  - Se al passo (3) si sta cercando di eliminare un dipendente assegnato ad uno o più turni per i giorni futuri, la cancellazione è annullata e viene visualizzato un messaggio di errore.

**3.3 UC3: Modifica dipendente**

**Descrizione** Si vogliono modificare alcuni dati di un dipendente.

**Requisiti coperti** DD03-DD08, DA01, DA08

**Attori coinvolti** Gestore del personale

**Precondizioni** Vari utenti sono presenti nel database.

**Postcondizioni** Alcuni dati del dipendente scelto sono modificati.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale clicca su “Lista dipendenti” nella pagina di gestione dei dipendenti.
  - (a) Viene visualizzata la lista di tutti i dipendenti.
2. Il gestore del personale seleziona il dipendente che vuole modificare.
3. Il gestore del personale clicca su “Modifica dipendente”.
  - (a) Viene mostrata la form per la modifica dei dati.
  - (b) Il gestore del personale modifica i dati.
4. Il gestore del personale clicca su “Conferma modifica”.
  - (a) E’ richiesta la conferma dell’azione: se l’azione è confermata, il dipendente è modificato.
  - (b) I nuovi dati del dipendente vengono salvati all’interno del database.
  - (c) Il dipendente riceve una e-mail contenente la conferma di inserimento ed il riepilogo delle informazioni inserite.
  - (d) Il sistema mostra un riepilogo indicante, per ogni attività disponibile, quali e quanti dipendenti sono disponibili.
  - (e) Viene aggiornata la lista di tutti i dipendenti, in modo che si possa eventualmente procedere ad una ulteriore modifica.

### Alternative

- Inserimento dati errati
  - Al passo (3.b) sono inseriti dati errati, in questo caso la modifica è annullata e viene lanciato un messaggio di errore.
  - Con “dati errati” si intende nome, cognome, posizione vuoti o formato delle date diverso da gg/mm/aaaa.
- Dipendente a tempo indeterminato
  - Al passo (3.b) non viene compilato il campo “data fine contratto”.
  - Quando si clicca “Conferma modifica” viene lanciato un messaggio di avviso per informare del campo vuoto, ma è comunque possibile proseguire.
- Attività non presente
  - Al passo (3.b) il gestore del personale non trova l’attività da assegnare al dipendente.
  - Il gestore del personale clicca sul pulsante “Nuova Attività”.

\* Da qua si procede come da UC4.

- Dipendente senza indirizzo e-mail
  - Al passo (3.b) non viene compilato il campo “indirizzo e-mail”.
  - Quando si clicca “Prosegui” viene lanciato un messaggio di avviso per informare del campo vuoto, ma è comunque possibile proseguire.

### 3.4 UC4: Inserimento nuova attività

**Descrizione** Si vuole inserire una nuova attività.

**Requisiti coperti** DA01, DA05, DA06, DA07, DA08

**Attori coinvolti** Gestore del personale

**Precondizioni** L’attività non è presente nel database.

**Postcondizioni** L’attività viene inserita nel database.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale preme il tasto “Aggiungi attività”.
2. Il gestore del personale compila la form in cui sono richiesti i dati dell’attività (codice, nome, postazione, descrizione).
3. Il gestore del personale preme “OK”
  - (a) I dati della nuova attività vengono inseriti nel database.
  - (b) Il sistema mostra un riepilogo indicante i dati delle attività attualmente presenti nel database.

#### Alternative

- Inserimento dati errati
  - Al passo (2) sono inseriti dati errati, in questo caso l’inserimento è annullato e viene lanciato un messaggio di errore.
  - Con “dati errati” si intende codice, nome, postazione o descrizione vuoti.

#### Estensioni

- Stampa dati di riepilogo
  - Al passo (3.b) il gestore del personale può cliccare sul pulsante “Stampa”, per stampare la scheda di riepilogo contenente i dati delle attività inserite.

### 3.5 UC5: Cancellazione di una attività

**Descrizione** Si vuole eliminare una attività precedentemente inserita.

**Requisiti coperti** DA02, DA05, DA06

**Attori coinvolti** Gestore del personale

**Precondizioni** Sono presenti varie attività nel database.

**Postcondizioni** L'attività viene cancellata dal database.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale preme il tasto "Lista attività".
  - (a) Viene visualizzata la lista di tutte le attività.
2. Il gestore del personale seleziona una attività.
3. Il gestore del personale clicca su "Elimina attività".
  - (a) E' richiesta la conferma dell'azione: se l'azione è confermata, l'attività è eliminata.
  - (b) Viene aggiornata la lista di tutte le attività, in modo che si possa eventualmente procedere ad una ulteriore eliminazione.

#### Alternative

- Attività assegnata a dipendenti
  - Al passo (3) si sta cercando di eliminare una attività assegnata ancora da almeno un dipendente, la cancellazione è annullata e viene visualizzato un messaggio di errore.
  - In questa situazione sarà necessario, in via preventiva, rimuovere l'attività dalla lista di quelle ricoperte dal dipendente, come da UC3.

### 3.6 UC6: Modifica di una attività

**Descrizione** Si vuole modificare un'attività precedentemente inserita.

**Requisiti coperti** DA03, DA05-DA08

**Attori coinvolti** Gestore del personale

**Precondizioni** Sono presenti varie attività nel database.

**Postcondizioni** L'attività viene modificata nel database.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale preme il tasto “Lista attività”.
  - (a) Viene visualizzata la lista di tutte le attività.
2. Il gestore del personale seleziona una attività.
3. Il gestore del personale clicca su "Modifica attività".
  - (a) Viene mostrata una form nella quale è possibile modificare i vari dati.
  - (b) Il gestore del personale modifica i dati.
4. Il gestore del personale clicca su "Conferma modifica".
  - (a) E' richiesta la conferma dell'azione: se l'azione è confermata, l'attività è modificata.
  - (b) I nuovi dati dell'attività sono inseriti all'interno del database.
  - (c) Il sistema mostra un riepilogo indicante i dati delle attività attualmente presenti nel database.

#### **Alternative**

- Inserimento dati errati
  - Al passo (3) sono inseriti dati errati, in questo caso l'inserimento è annullato e viene lanciato un messaggio di errore.
  - Con “dati errati” si intende codice, nome, postazione o descrizione vuoti.

#### **Estensioni**

- Stampa dati di riepilogo
  - Al passo (4.c), il gestore del personale può cliccare sul pulsante “Stampa”, per stampare la scheda di riepilogo contenente i dati delle attività inserite.

### **3.7 UC7. Allocazione dipendenti per ogni attività**

**Descrizione** Si vuole impostare il numero di dipendenti necessari per ogni attività in un intervallo di date.

**Requisiti coperti** DA04-DA08

**Attori coinvolti** Gestore del personale

**Precondizioni** Sono presenti varie attività nel database.

**Postcondizioni** Il numero impostato di dipendenti necessari per ogni attività viene inserito all'interno del database.

**Processo** Di seguito è descritto il processo:

1. Il gestore del personale preme il tasto “Imposta richiesta manodopera per attività”.
2. Il gestore del personale seleziona l’intervallo di date all’interno del quale vuole impostare la richiesta di manodopera necessaria per attività.
3. Il gestore del personale seleziona l’attività.
4. Il gestore del personale seleziona il turno per il quale vuole impostare la richiesta di manodopera.
5. Il gestore del personale seleziona il numero di dipendenti, che eseguono l’attività richiesta, necessari per il turno selezionato.
6. Il gestore del personale clicca sul pulsante “Conferma”.
  - (a) I nuovi dati vengono inseriti all’interno del database.
  - (b) Il sistema mostra un riepilogo indicante i dati delle attività attualmente presenti nel database, con relative richieste di dipendenti, per l’intervallo di date selezionato.

**Alternative**

- Inserimento data errata
  - Al passo (2) il gestore del personale inserisce delle date non valide o con formato non corretto. Viene mostrato un messaggio di errore e viene data la possibilità di modificare le due date.
- Inserimento numero dipendenti errato
  - Al passo (5) il gestore del personale inserisce un valore numerico negativo. Viene mostrato un messaggio di errore e viene data la possibilità di modificare il valore.

### 3.8 UC8: Generazione turnazione settimanale

**Descrizione** Si vuole generare la turnazione di una nuova settimana, includendo tutti i dipendenti che non sono in ferie.

L’assegnazione dei turni deve rispettare il numero massimo di ore settimanali di ciascun dipendente.

**Requisiti coperti** TD01, TD02, TD05, TD06, TD07, TD08, TD12, TD15, DA04-DA08

**Attori coinvolti** Impiegato dell’ufficio amministrazione

**Precondizioni**

- I dipendenti sono inseriti all’interno del database del programma.
- Non è ancora stata generata una turnazione per la settimana.

**Postcondizioni** Turni assegnati a tutti i dipendenti per la settimana richiesta, mail (contenente il resoconto della turnazione generata) inviata a ciascun dipendente.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione preme il pulsante "Genera turnazione settimanale".
2. L'impiegato dell'ufficio amministrazione seleziona la settimana per la quale vuole generare la turnazione.
3. L'impiegato dell'ufficio amministrazione preme il pulsante "Genera".
  - (a) Il sistema genera in automatico le turnazioni.
  - (b) Il sistema mostra le turnazioni generate all'impiegato dell'ufficio amministrazione.
4. L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma".
  - (a) Le turnazioni assegnate vengono inserite all'interno del database.
  - (b) Ciascun dipendente riceve una mail contenente le indicazioni sui turni che gli sono stati assegnati per la settimana.

#### Alternative

- Selezione settimana errata
  - Al passo (2) l'impiegato dell'ufficio amministrazione seleziona una settimana non esistente o per la quale è già stata assegnata una turnazione. Viene quindi mostrato un messaggio di errore.
- Dipendenti non sufficienti
  - Al passo (3) il sistema non riesce a trovare un'allocatione ammissibile, poichè i dipendenti o la disponibilità di ore non sono sufficienti. Viene quindi mostrato un messaggio di errore.
- Indirizzi e-mail non inseriti
  - Al passo (4) il sistema non trova gli indirizzi e-mail di alcuni dipendenti. Viene mostrata una finestra che comunica i dati mancanti.

#### Estensioni

- Generazione nuova turnazione
  - Al passo (4) l'impiegato dell'ufficio amministrazione clicca su "Genera nuova turnazione". Si riparte quindi dal passo (3.a).

### 3.9 UC9: Visualizzazione turnazione settimanale

**Descrizione** Si vuole visualizzare la turnazione assegnata per una specifica settimana.



**Requisiti coperti** TD01, TD02, TD03, TD05, TD13, TD14

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni** La turnazione della settimana è inserita all'interno del database, quindi è già stata generata come da UC8.

**Postcondizioni** La turnazione della settimana viene mostrata all'impiegato dell'ufficio amministrazione

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione preme il pulsante "Visualizza turnazione settimanale".
2. L'impiegato dell'ufficio amministrazione seleziona la settimana per la quale vuole visualizzare la turnazione.
3. L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma".
  - (a) La turnazione della settimana selezionata viene mostrata.

#### Alternative

- Selezione settimana errata
  - Al passo (2) l'impiegato dell'ufficio amministrazione seleziona una settimana non esistente o per la quale non è ancora stata generata una turnazione. Viene quindi mostrato un messaggio di errore.

#### Estensioni

- Visualizzazione turni del singolo dipendente
  - Al passo (2) l'impiegato dell'ufficio amministrazione può anche selezionare il dipendente del quale vuole conoscere la turnazione. Al passo (3), quindi, non verranno mostrati tutti gli orari, ma solamente quelli del dipendente selezionato.
- Invio mail aggiornamento
  - Al passo (3) l'impiegato dell'ufficio amministrazione può anche cliccare sul pulsante "Invia mail". Il sistema si occuperà, quindi, di inviare una mail a tutti i dipendenti contenente i relativi turni.
- Stampa report allocazione dipendenti sui turni
  - Al passo (3) l'impiegato dell'ufficio amministrazione può anche cliccare sul pulsante "Stampa turnazione". Il sistema si occuperà, quindi, di esportare in formato PDF la tabella con l'allocazione dei dipendenti sui turni, per la settimana selezionata.

### 3.10 UC10: Visualizzazione turnazione giornaliera

**Descrizione** Si vuole visualizzare la turnazione assegnata per uno specifico giorno.

**Requisiti coperti** TD02, TD04, TD05, TD14

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni** La turnazione del giorno è inserita all'interno del database, quindi è già stata generata come da UC1.

**Postcondizioni** La turnazione del giorno viene mostrata all'impiegato dell'ufficio amministrazione

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione preme il pulsante "Visualizza turnazione giornaliera".
2. L'impiegato dell'ufficio amministrazione seleziona il giorno per il quale vuole visualizzare la turnazione.
3. L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma".
  - (a) La turnazione del giorno selezionato viene mostrata.

#### Alternative

- Selezione giorno errata
  - Al passo (2) l'impiegato dell'ufficio amministrazione seleziona una data non esistente o per la quale non è ancora stata generata una turnazione. Viene quindi mostrato un messaggio di errore.

#### Estensioni

- Visualizzazione turni del singolo dipendente
  - Al passo (2) l'impiegato dell'ufficio amministrazione può anche selezionare il dipendente del quale vuole conoscere la turnazione. Al passo (3), quindi, non verranno mostrati tutti gli orari, ma solamente quelli del dipendente selezionato.
- Invio mail aggiornamento
  - Al passo (3) l'impiegato dell'ufficio amministrazione può anche cliccare sul pulsante "Invia mail". Il sistema si occuperà, quindi, di inviare una mail a tutti i dipendenti contenenti i relativi turni per il giorno visualizzato.
- Stampa report allocazione dipendenti sui turni

- Al passo (3) l'impiegato dell'ufficio amministrazione può anche cliccare sul pulsante "Stampa turnazione". Il sistema si occuperà, quindi, di esportare in formato PDF la tabella con l'allocazione dei dipendenti sui turni, per il giorno selezionato.

### 3.11 UC11: Cambio turno dipendente

**Descrizione** Si vuole, una volta ricevuta una specifica richiesta da un dipendente, modificare il relativo turno. La modifica di questo turno dovrà comportare, eventualmente, la modifica del turno di altri dipendenti.

**Requisiti coperti** TD01, TD02, TD09, TD10, TD11, TD12, TD15

**Attori coinvolti** Impiegato dell'ufficio amministrazione

#### Precondizioni

- I dipendenti sono inseriti all'interno del database del programma.
- E' già stata generata una turnazione per la giornata richiesta.

#### Postcondizioni

- Turni modificati, generando un assegnamento consistente.
- Mail contenente i nuovi turni inviata ai dipendenti che hanno subito variazioni.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione preme il pulsante "Modifica turno dipendente".
2. L'impiegato dell'ufficio amministrazione seleziona il giorno per il quale vuole modificare la turnazione.
3. L'impiegato dell'ufficio amministrazione seleziona il dipendente per il quale vuole modificare la turnazione.
4. L'impiegato dell'ufficio amministrazione seleziona il nuovo turno da assegnare al dipendente.
5. L'impiegato dell'ufficio amministrazione preme il pulsante "Genera".
  - (a) Il sistema modifica in automatico le turnazioni.
  - (b) Il sistema mostra le turnazioni aggiornate all'impiegato dell'ufficio amministrazione.
6. L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma".
  - (a) Le turnazioni aggiornate vengono inserite all'interno del database.
  - (b) Ciascun dipendente che ha subito modifiche riceve una mail contenente le indicazioni sulle modifiche subite ai propri orari.

### Alternative

- Selezione giorno errata
  - Al passo (2) l'impiegato dell'ufficio amministrazione seleziona una data non esistente o per la quale non è ancora stata generata una turnazione. Viene quindi mostrato un messaggio di errore.
- Selezione dipendente errato
  - Al passo (3) l'impiegato dell'ufficio amministrazione seleziona un dipendente non esistente. Viene quindi mostrato un messaggio di errore.
- Dipendenti non sufficienti
  - Al passo (4) il sistema non riesce a trovare un'allocazione ammissibile, poichè i dipendenti o la disponibilità di ore non sono sufficienti. Viene quindi mostrato un messaggio di errore.
- Indirizzi e-mail non inseriti
  - Al passo (5) il sistema non trova gli indirizzi e-mail di alcuni dipendenti. Viene mostrata una finestra che comunica i dati mancanti.

### Estensioni

- Generazione nuova turnazione
  - Al passo (6) l'impiegato dell'ufficio amministrazione clicca su "Genera nuova turnazione". Si riparte quindi dal passo (5.a).

## 3.12 UC12: Gestione dei dati provenienti dalla timbratrice

**Descrizione** Si vogliono estrarre i dati contenuti nei file XML, selezionando solamente i dati di interesse. Il gestore del personale può fare delle modifiche o eliminare alcuni dati.

Successivamente, confermando i dati, gli orari delle timbrature vengono aggregati per ogni dipendente, in modo da avere tutti gli orari di ingresso ed uscita del giorno.

**Requisiti coperti** FM00-FM05, SM09, SM10, SM20-SM23

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni** E' disponibile il file .XML contenente i dati delle timbrature. Questo file contiene i dati "grezzi", quindi non ancora elaborati.

**Postcondizioni**

- Le informazioni di interesse (orario, ID badge, tipo timbratura) vengono estratte e memorizzate all'interno del database.
- Nel caso si siano modificati alcuni dati, nel database sono presenti i dati aggiornati.
- Nel caso si siano eliminati alcuni dati, nel database non sono più presenti tali dati.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Importa file .XML".
  - (a) Viene mostrata una maschera per la selezione del file da importare.
2. L'impiegato dell'ufficio amministrazione seleziona il file che vuole importare.
3. Il sistema estrae i vari record delle timbrature tramite parsing, scorrendo i vari tag.
  - (a) Viene mostrata una form contenente i dati importati, per un'eventuale modifica.
4. L'impiegato dell'ufficio amministrazione può fare una delle seguenti scelte:
  - (a) Modifica dati:
    - i. L'impiegato dell'ufficio amministrazione seleziona una timbratura che vuole modificare.
    - ii. L'impiegato dell'ufficio amministrazione effettua un doppio click sul dato che vuole modificare.
      - I campi con i dati della timbratura diventano modificabili.
    - iii. L'impiegato dell'ufficio amministrazione inserisce i dati corretti.
    - iv. L'impiegato dell'ufficio amministrazione preme il pulsante INVIO.
      - Viene aggiornato il contenuto della form di revisione dei dati importati.
      - Si ritorna al passo (4).
  - (b) Eliminazione dei dati:
    - i. L'impiegato dell'ufficio amministrazione seleziona una timbratura che vuole eliminare.
    - ii. L'impiegato dell'ufficio amministrazione svuota tutti i campi della riga che vuole eliminare.
      - Viene aggiornato il contenuto della form di revisione dei dati importati.
      - Si ritorna al passo (4).
5. L'impiegato dell'ufficio amministrazione conferma i dati, tramite il pulsante "Salva".

- (a) Il sistema salva i dati grezzi, provenienti dalla timbratrice, con le eventuali modifiche effettuate manualmente dall'impiegato dell'ufficio amministrazione.
- (b) Il sistema aggrega i dati di ogni dipendente, grazie al codice identificativo, al fine di avere per ogni dipendente tutti gli orari di ingresso/uscita giornalieri.
- (c) I dati aggregati vengono inseriti all'interno del database.

### Alternative

- Struttura file .XML errata
  - Al passo (2) l'impiegato dell'ufficio amministrazione seleziona un file che non soddisfa la struttura prevista (ad esempio con tag mancante o tag aggiuntivo). Il processo di parsing viene quindi interrotto e viene visualizzato un messaggio di errore.
- Nessun file .XML selezionato
  - Al passo (2) l'impiegato dell'ufficio amministrazione non seleziona alcun file, oppure seleziona un file non .XML. Il processo di parsing viene quindi interrotto e viene visualizzato un messaggio di errore.
- Inserimento di una data errata
  - Al passo (4.a.iii) l'impiegato dell'ufficio amministrazione inserisce un valore errato per il campo "Data". Viene quindi mostrato un messaggio di errore e viene data la possibilità di modificare il campo errato.
- Inserimento di un orario errato
  - Al passo (4.a.iii) l'impiegato dell'ufficio amministrazione inserisce un valore errato per il campo "Orario Timbratura". Viene quindi mostrato un messaggio di errore e viene data la possibilità di modificare il campo errato.

### 3.13 UC13: Gestione straordinari

**Descrizione** Si vogliono inserire/aggiornare o eliminare i dati relativi alle ore straordinarie necessarie in un determinato periodo.

**Requisiti coperti** SM17-SM19, SM26

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni** E' valida una di queste precondizioni:

- Non è ancora stato inserito alcun dato riguardante le ore di straordinario necessarie.
- I dati sulle ore di straordinario necessarie sono già stati inseriti e devono essere aggiornati.

- I dati sulle ore di straordinario necessarie sono già stati inseriti e devono essere eliminati.

**Postcondizioni** In base all'azione eseguita è valida almeno una delle seguenti postcondizioni:

- I nuovi dati riguardanti le ore di straordinario sono inseriti all'interno del database.
- I dati riguardanti le ore di straordinario già presenti nel database sono stati aggiornati.
- I dati riguardanti le ore di straordinario già presenti nel database sono stati eliminati.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Gestisci fabbisogno straordinari".
2. L'impiegato dell'ufficio amministrazione ha a disposizione tre scelte:
  - (a) Può cliccare sul pulsante "Inserisci".
  - (b) Può cliccare sul pulsante "Modifica".
  - (c) Può cliccare sul pulsante "Elimina".
3. Se l'impiegato dell'ufficio amministrazione ha cliccato su "Inserisci":
  - (a) L'impiegato dell'ufficio amministrazione inserisce la data per la quale vuole inserire le ore di straordinario necessarie.
  - (b) L'impiegato dell'ufficio amministrazione inserisce l'attività per la quale sono necessarie le ore di straordinario.
  - (c) L'impiegato dell'ufficio amministrazione inserisce il numero di ore necessarie.
  - (d) L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma" per confermare l'inserimento.
    - i. I dati vengono inseriti nel database.
4. Se l'impiegato dell'ufficio amministrazione ha cliccato su "Modifica":
  - (a) L'impiegato dell'ufficio amministrazione inserisce la data per la quale vuole modificare le ore di straordinario necessarie.
  - (b) L'impiegato dell'ufficio amministrazione inserisce l'attività per la quale vuole modificare le ore di straordinario necessarie.
    - i. Il campo contenente le ore di straordinario viene pre-compilato, indicando i dati già inseriti nel database.
  - (c) L'impiegato dell'ufficio amministrazione inserisce il nuovo numero di ore necessarie.
  - (d) L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma" per confermare l'aggiornamento.

- i. I dati vengono aggiornati nel database.
- 5. Se l'impiegato dell'ufficio amministrazione ha cliccato su "Elimina":
  - (a) L'impiegato dell'ufficio amministrazione inserisce la data per la quale vuole eliminare le ore di straordinario necessarie.
  - (b) L'impiegato dell'ufficio amministrazione inserisce l'attività per la quale vuole eliminare le ore di straordinario necessarie.
  - (c) L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma eliminazione" per confermare la cancellazione.
    - i. I dati vengono eliminati dal database.

### Alternative

- Inserimento dati errati
  - Ai passi (3.\*), (4.\*) o (5.\*) vengono inseriti dei dati errati. L'operazione viene annullata e viene mostrato un messaggio di errore.
  - Per dati errati si intende:
    - \* Date non corrette.
    - \* Attività non esistenti.
    - \* Numero di ore non corretto (*ad esempio*  $< 0$ ).
    - \* Qualsiasi dato che non rispetta vincoli di dominio (*ad esempio superamento del massimale di ore di straordinario possibili in una giornata*).
- Modifica di una tupla non presente
  - Al passo (4.b) l'impiegato dell'ufficio amministrazione tenta di modificare una tupla non presente. Viene mostrato un messaggio di errore e viene data la possibilità di modificare i dati inseriti.
- Eliminazione di una tupla non presente
  - Al passo (5.c) l'impiegato dell'ufficio amministrazione tenta di eliminare una tupla non presente. Viene mostrato un messaggio di errore e viene data la possibilità di modificare i dati inseriti.

### 3.14 UC14: Gestione giorni di malattia/ferie

**Descrizione** Si vogliono inserire/aggiornare o eliminare i dati relativi ai giorni di malattia o di ferie comunicati dai dipendenti.

**Requisiti coperti** SM09, SM14-SM16

**Attori coinvolti** Impiegato dell'ufficio amministrazione



**Precondizioni** E' valida una di queste precondizioni:

- Non è ancora stato inserito alcun dato riguardante i giorni di malattia o di ferie per un dipendente.
- I dati sui giorni di malattia o ferie di un dipendente sono già stati inseriti e devono essere aggiornati.
- I dati sui giorni di malattia o ferie di un dipendente sono già stati inseriti e devono essere eliminati.

**Postcondizioni** In base all'azione eseguita è valida almeno una delle seguenti postcondizioni:

- I nuovi dati riguardanti i giorni di malattia o ferie sono inseriti all'interno del database.
- I dati riguardanti i giorni di malattia o ferie già presenti nel database sono stati aggiornati.
- I dati riguardanti i giorni di malattia o ferie già presenti nel database sono stati eliminati.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Gestisci giorni di ferie"/"Gestisci giorni di malattia".
2. L'impiegato dell'ufficio amministrazione ha a disposizione tre scelte:
  - (a) Può cliccare sul pulsante "Inserisci".
  - (b) Può cliccare sul pulsante "Modifica".
  - (c) Può cliccare sul pulsante "Elimina".
3. Se l'impiegato dell'ufficio amministrazione ha cliccato su "Inserisci":
  - (a) L'impiegato dell'ufficio amministrazione inserisce l'identificativo del dipendente in ferie/malattia.
  - (b) L'impiegato dell'ufficio amministrazione inserisce l'intervallo di date per il quale il dipendente è in ferie/malattia.
  - (c) L'impiegato dell'ufficio amministrazione preme il pulsante "Conferma" per confermare l'inserimento.
    - i. I dati vengono inseriti nel database.
4. Se l'impiegato dell'ufficio amministrazione ha cliccato su "Modifica":
  - (a) L'impiegato dell'ufficio amministrazione inserisce l'identificativo del dipendente in ferie/malattia.
  - (b) L'impiegato dell'ufficio amministrazione imposta l'intervallo di date contenente i giorni da modificare.
    - i. Vengono mostrati i dati del dipendente per il mese selezionato.

- (c) L'impiegato dell'ufficio amministrazione modifica lo stato di uno o più giorni (ferie-no ferie / malattia-no malattia) e clicca sul pulsante "Conferma".
  - i. I dati vengono aggiornati nel database.
- 5. Se l'impiegato dell'ufficio amministrazione ha cliccato su "Elimina":
  - (a) L'impiegato dell'ufficio amministrazione inserisce l'identificativo del dipendente in ferie/malattia.
  - (b) L'impiegato dell'ufficio amministrazione imposta l'intervallo di date contenente i giorni da eliminare.
    - i. Vengono mostrati i dati del dipendente per il mese selezionato.
  - (c) L'impiegato dell'ufficio amministrazione elimina uno o più giorni e clicca sul pulsante "Conferma".
    - i. I dati vengono eliminati dal database.

### Alternative

- Inserimento dati errati
  - Ai passi (3.\*), (4.\*) o (5.\*) vengono inseriti dei dati errati. L'operazione viene annullata e viene mostrato un messaggio di errore.
  - Per dati errati si intende:
    - \* Date non corrette.
    - \* Identificativo dipendente non esistente.
    - \* Qualsiasi dato che non rispetta vincoli di dominio.
- Inserimento di dati incompatibili
  - Ai passi (3.\*), (4.\*) o (5.\*) vengono inseriti dei dati incompatibili. L'operazione viene annullata e viene mostrato un messaggio di errore.
  - Per dati incompatibili si intende il caso in cui vengano inserite una o più giornate di ferie/malattia per un dipendente che ha già superato il monte ore disponibile.
- Aggiornamento di una tupla non presente
  - Al passo (4.c) l'impiegato dell'ufficio amministrazione tenta di aggiornare una tupla non presente. Viene mostrato un messaggio di errore e viene data la possibilità di modificare i dati inseriti.
- Eliminazione di una tupla non presente
  - Al passo (5.c) l'impiegato dell'ufficio amministrazione tenta di eliminare una tupla non presente. Viene mostrato un messaggio di errore e viene data la possibilità di modificare i dati inseriti.

### 3.15 UC15: Visualizzazione e calcolo giornaliero ore di lavoro

**Descrizione** Si vogliono calcolare le ore di lavoro relative ai dipendenti, a partire dai dati derivanti dal parsing del file .XML (si veda UC12).

**Requisiti coperti** SM00-SM02, SM04, SM25, SM27

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni**

- I dati relativi agli orari giornalieri dei dipendenti non sono presenti.
- I dati grezzi (con eventuale modifica manuale o revisione) sono già stati importati dal file .XML (si veda UC12).

**Postcondizioni** Per ogni dipendente sono calcolate le ore giornaliere di lavoro ordinario, straordinario, di ferie, di permesso e malattia.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Visualizza orari dipendenti".
2. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Visualizza orari giornalieri".
  - (a) Viene richiesto all'impiegato dell'ufficio amministrazione di inserire la data di interesse.
3. L'impiegato dell'ufficio amministrazione inserisce la data di interesse e preme sul pulsante "Conferma" per avviare il processo.
  - (a) Viene avviato il processo di calcolo degli orari del giorno selezionato:
    - Lavoro ordinario: se gli orari rientrano nelle fasce di lavoro ordinario, in base al turno assegnato a ciascun dipendente.
    - Lavoro straordinario: se un dipendente supera gli orari di lavoro ordinario.
    - Ferie: se il giorno d'interesse è stato inserito, per il dipendente considerato, come giorno di ferie (si veda UC14).
    - Permesso: se le ore di ordinario sono inferiori a quelle giornaliere contrattuali.
    - Malattia: se il giorno d'interesse è stato inserito, per il dipendente considerato, come giorno di malattia (si veda UC14).
  - (b) Viene mostrata una maschera riassuntiva in cui, per ogni dipendente, è indicato il numero di ore giornaliere di lavoro ordinario, straordinario, di ferie, di permesso e di malattia.

**Alternative**

- Dipendente in ferie presente
  - Al passo (3), il sistema rileva una timbratura per un dipendente che è considerato in ferie, in base ai dati inseriti. Viene, quindi, mostrato un messaggio di errore informativo.

- Dipendente in malattia presente
  - Al passo (3), il sistema rileva una timbratura per un dipendente che è considerato in malattia, in base ai dati inseriti. Viene, quindi, mostrato un messaggio di errore informativo.

### Estensioni

- Modifica orari timbratura
  - Al passo (3.b), l'impiegato dell'ufficio amministrazione può modificare gli orari di ingresso e uscita del dipendente.
  - In questo caso, una volta confermata la modifica, viene ripetuto il calcolo delle ore.

### 3.16 UC16: Visualizzazione e calcolo mensile ore di lavoro

**Descrizione** Si vogliono calcolare le ore di lavoro relative ai dipendenti, a partire dai dati derivanti dal parsing del file .XML (si veda UC12).

**Requisiti coperti** SM05-SM08, SM13, SM24, SM25

**Attori coinvolti** Impiegato dell'ufficio amministrazione

#### Precondizioni

- I dati relativi agli orari mensili dei dipendenti non sono presenti.
- I dati grezzi (con eventuale modifica manuale o revisione) sono già stati importati dal file .XML (si veda UC12).

**Postcondizioni** Per ogni dipendente sono calcolate le ore mensili di lavoro ordinario, straordinario, di ferie, di permesso e malattia.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Visualizza orari dipendenti".
2. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Visualizza orari mensili".
  - (a) Viene richiesto all'impiegato dell'ufficio amministrazione di inserire il mese e l'anno di interesse.
  - (b) Viene richiesto all'impiegato dell'ufficio amministrazione di inserire l'identificativo del dipendente per il quale vuole conoscere gli orari mensili.
3. L'impiegato dell'ufficio amministrazione inserisce l'anno ed il mese di interesse.

4. L'impiegato dell'ufficio amministrazione inserisce l'identificativo del dipendente per il quale vuole conoscere gli orari mensili e preme sul pulsante "Conferma" per avviare il processo.
- (a) Viene avviato il processo di calcolo degli orari del mese selezionato:
    - Lavoro ordinario: se gli orari rientrano nelle fasce di lavoro ordinario, in base al turno assegnato ogni giorno al dipendente.
    - Lavoro straordinario: se il dipendente supera gli orari di lavoro ordinario.
    - Ferie: se un determinato giorno è stato inserito, per il dipendente considerato, come giorno di ferie (si veda UC14).
    - Permesso: se le ore di ordinario sono inferiori a quelle giornaliere contrattuali.
    - Malattia: se un determinato giorno è stato inserito, per il dipendente considerato, come giorno di malattia (si veda UC14).
  - (b) Viene mostrata una maschera riassuntiva in cui, per il dipendente desiderato, è indicato il numero di ore mensili di lavoro ordinario, straordinario, di ferie, di permesso e di malattia.

### Alternative

- Dipendente in ferie presente
  - Al passo (4), il sistema rileva una timbratura per il dipendente in un giorno che è stato precedentemente inserito come giorno di ferie. Viene, quindi, mostrato un messaggio di errore informativo.
- Dipendente in malattia presente
  - Al passo (4), il sistema rileva una timbratura per il dipendente in un giorno che è stato precedentemente inserito come giorno di malattia. Viene, quindi, mostrato un messaggio di errore informativo.
- Superamento limite di lavoro straordinario
  - Al passo (4), il sistema rileva una quantità di ore di lavoro straordinario superiore al limite. In questo caso viene mostrato un messaggio di errore.

### Estensioni

- Stampa report mensile dipendente
  - Al passo (4), l'impiegato dell'ufficio amministrazione può cliccare sul pulsante "Stampa" per esportare un PDF contenente il report mensile degli orari del dipendente.
  - Si procede come da UC19.
- Modifica orari timbratura
  - Al passo (3.b), l'impiegato dell'ufficio amministrazione può modificare gli orari di ingresso e uscita del dipendente.
  - In questo caso, una volta confermata la modifica, viene ripetuto il calcolo delle ore.

### 3.17 UC17: Applicazione incremento percentuale straordinario

**Descrizione** Si vuole applicare un incremento percentuale in caso di lavoro straordinario o di lavoro notturno (prima delle 08:00 e dopo le 22:00).

**Requisiti coperti** SM11

**Attori coinvolti** Gestione degli orari dei dipendenti

**Precondizioni** Ad ogni orario del dipendente è applicata la paga base oraria.

**Postcondizioni** Viene applicato l'incremento percentuale sia alle ore di lavoro straordinario che in quelle di lavoro ordinario.

**Processo** Di seguito è descritto il processo:

1. Il sistema seleziona dal database le ore di lavoro straordinario o notturno.
2. Il sistema applica l'incremento percentuale alle ore notturne.
3. Il sistema applica l'incremento percentuale alle ore di straordinario.
4. I dati vengono memorizzati all'interno del database.

### 3.18 UC18: Calcolo stipendio

**Descrizione** Si vuole calcolare lo stipendio mensile di ogni dipendente.

**Requisiti coperti** SM12

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni** Sono caricate le timbrature di ogni dipendente.

**Postcondizioni** Per ogni dipendente è calcolato lo stipendio mensile.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Calcola stipendi".
  - (a) Viene mostrata una form con la possibilità di selezionare il mese e l'anno di interesse.
2. L'impiegato dell'ufficio amministrazione seleziona l'anno ed il mese di interesse.
3. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Calcola".
  - (a) Il sistema calcola in automatico gli stipendi per tutti i dipendenti.

- (b) Nel caso di straordinari o orari notturni, viene applicato un incremento percentuale come da UC17.
- (c) Il sistema applica eventuali tassazioni.
- (d) Viene mostrata una form con l'indicazione, per ogni dipendente, del relativo stipendio.

### Alternative

- Inserimento anno o mese errato
  - Al passo (2), l'impiegato dell'ufficio amministrazione inserisce dei valori errati per l'anno o il mese. Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.
- Inserimento anno o mese futuro
  - Al passo (2), l'impiegato dell'ufficio amministrazione inserisce dei valori futuri per l'anno o il mese (*quindi per i quali ancora non si hanno indicazioni sulle timbrature*). Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.

## 3.19 UC19: Generazione report dipendente

**Descrizione** Si vuole stampare un report, per un dipendente selezionato, che rispecchia la situazione mensile lavorativa.

**Requisiti coperti** SM03, SM13

**Attori coinvolti** Impiegato dell'ufficio amministrazione

**Precondizioni** Sono già stati caricati i dati di ore di ordinario, straordinario, ferie, permessi e malattia per il dipendente.

**Postcondizioni** Viene prodotto un report mensile del dipendente, con statistiche riguardanti le varie tipologie di lavoro svolto, resoconto delle timbrature e stipendio.

**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Stampa report dipendente".
  - (a) Viene mostrata una form con la possibilità di selezionare il mese e l'anno di interesse, oltre che l'identificativo del dipendente per il quale si vuole ottenere il report.
2. L'impiegato dell'ufficio amministrazione seleziona l'anno ed il mese di interesse.

3. L'impiegato dell'ufficio amministrazione seleziona l'identificativo del dipendente di interesse.
4. L'impiegato dell'ufficio amministrazione clicca sul pulsante "Genera".
  - (a) Il sistema aggrega tutte le informazioni necessarie e produce il report PDF per il dipendente selezionato.

### Alternative

- Inserimento anno o mese errato
  - Al passo (2), l'impiegato dell'ufficio amministrazione inserisce dei valori errati per l'anno o il mese. Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.
- Inserimento anno o mese futuro
  - Al passo (2), l'impiegato dell'ufficio amministrazione inserisce dei valori futuri per l'anno o il mese (*quindi per i quali ancora non si hanno indicazioni sulle timbrature*). Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.
- Inserimento identificativo dipendente errato
  - Al passo (3), l'impiegato dell'ufficio amministrazione inserisce un identificativo di un dipendente non esistente. Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.

## 3.20 UC20: Prenotazione per straordinari

**Descrizione** Si vuole fare in modo che un dipendente possa prenotarsi per un certo numero di ore di straordinario previste.

**Requisiti coperti** SM28

**Attori coinvolti** Dipendente

**Precondizioni** Sono già stati caricati i dati relativi alle ore di straordinario necessarie, previste dall'azienda.

**Postcondizioni** Nel database è registrata la prenotazione del dipendente per un certo numero di ore di straordinario previste.



**Processo** Di seguito è descritto il processo:

1. Il dipendente clicca su "Prenotazione straordinari".
  - (a) Viene mostrata una form con l'elenco degli straordinari previsti (si veda UC13), per i quali non sono già terminate le prenotazioni. Si dà inoltre la possibilità di filtrare le date, inserendo l'intervallo di date desiderato.
2. Il dipendente seleziona la riga relativa alla data di interesse.
3. Il dipendente clicca su "Prenota".
4. Il dipendente inserisce il numero di ore per cui si prenota.
  - (a) Il dato inserito viene memorizzato nel database.
  - (b) Viene mostrato al dipendente un messaggio di conferma.

### Alternative

- Inserimento date errate
  - Al passo (2.a), il dipendente inserisce, per filtrare i risultati, delle date errate. Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.
- Inserimento ore sopra il limite
  - Al passo (4), il dipendente inserisce un numero di ore superiori a quelle disponibili. Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.
  - Lo stesso comportamento si ha quando il dipendente si prenota, nella stessa settimana o nello stesso mese, per un numero di ore che eccede quelle permesse legalmente.
- Prenotazione per date passate
  - Al passo (3), il dipendente cerca di prenotarsi per una data passata. Viene, quindi, mostrato un messaggio di errore.

### Estensioni

- Selezione date per filtro
  - Al passo (2.a) il dipendente può inserire una o due date, per filtrare quelle memorizzate.
  - In questo caso vengono mostrate solamente le date con straordinari disponibili, comprese all'interno dell'intervallo.

## 3.21 UC21: Verifica copertura turni assegnati

**Descrizione** Si vuole verificare che un dipendente abbia coperto tutti i turni che gli sono stati assegnati.

**Requisiti coperti** SM25**Attori coinvolti** Impiegato ufficio amministrazione**Precondizioni** Sono già stati caricati i turni per le date che si vogliono analizzare. Inoltre deve già essere caricato anche il dipendente all'interno del database.**Postcondizioni** Viene mostrata una finestra riepilogativa, indicante quali turni sono stati effettivamente coperti dal dipendente, e quali no.**Processo** Di seguito è descritto il processo:

1. L'impiegato dell'ufficio amministrazione clicca su "Verifica copertura turni".
  - (a) Viene mostrata una form nella quale l'impiegato dell'ufficio amministrazione può selezionare il dipendente e l'intervallo di interesse.
2. L'impiegato dell'ufficio amministrazione inserisce il dipendente per il quale vuole avviare il processo di verifica.
3. L'impiegato dell'ufficio amministrazione inserisce l'intervallo per il quale vuole avviare il processo di verifica.
4. L'impiegato dell'ufficio amministrazione clicca su "Verifica".
  - (a) Viene mostrata una form nella quale vengono mostrati i turni che sono stati effettivamente coperti dal dipendente.

**Alternative**

- Inserimento date errate
  - Al passo (3), l'impiegato dell'ufficio amministrazione inserisce, per filtrare i risultati, delle date errate. Viene, quindi, mostrato un messaggio di errore e viene data la possibilità di correggere i valori inseriti.

## 4 Use case diagram

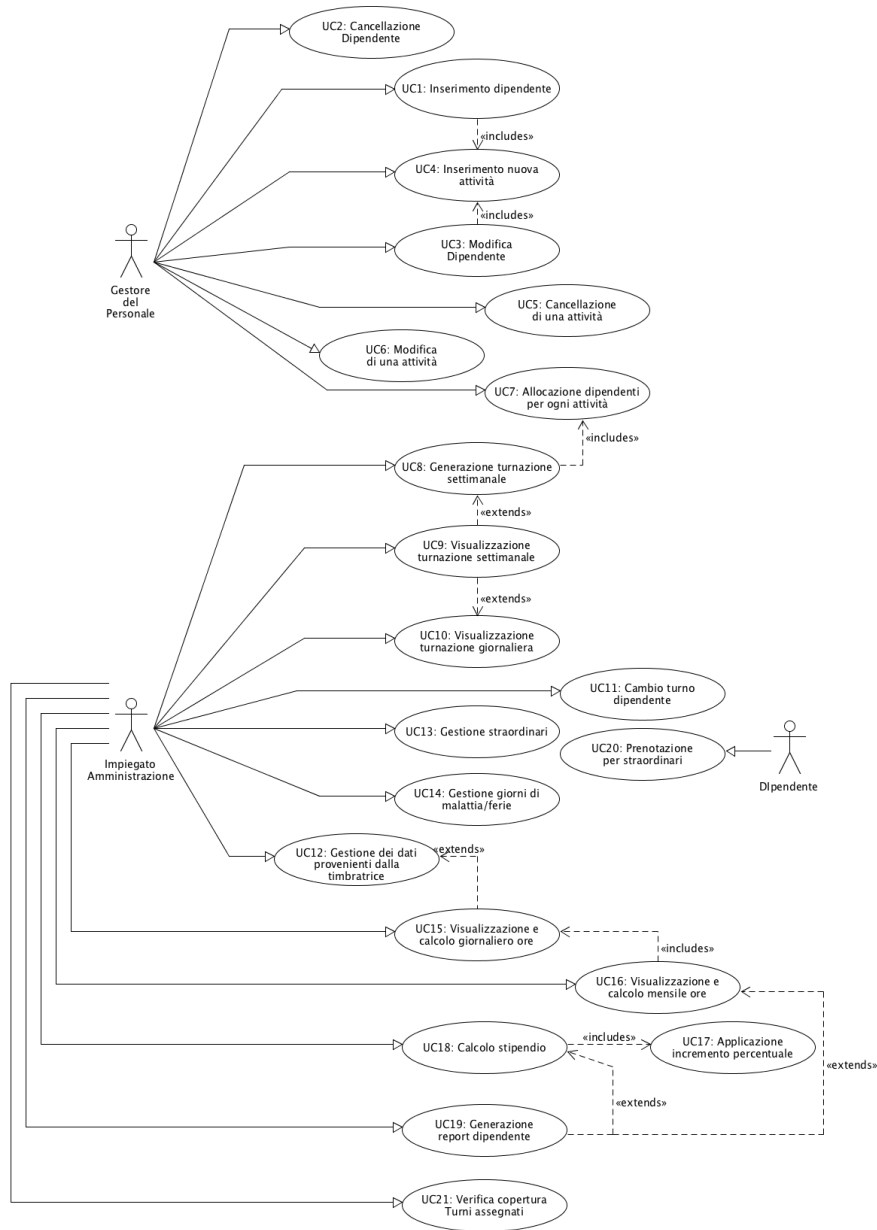


Figura 2.1: Use case diagram



## Capitolo 3

# Architettura

### 1 Deployment diagram

#### 1.1 Architettura hardware

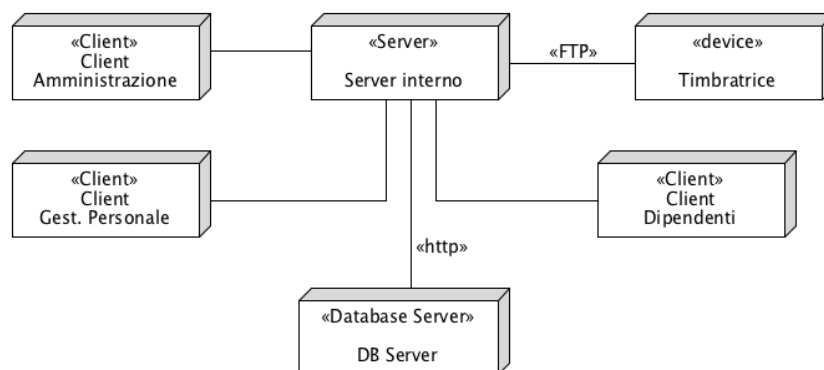


Figura 3.1: Architettura hardware

La figura 3.1 rappresenta l'architettura finale prevista per la realizzazione del sistema richiesto dal cliente. Attualmente, la dotazione hardware del cliente, è composta da:

- Un server interno in cui vengono memorizzati i file .XML prodotti dalla timbratrice, oltre ad altre informazioni non rilevanti per il progetto commissionato.
- Client differenti (ufficio amministrazione, ufficio gestione personale e dipendenti), suddivisi in base alla loro collocazione all'interno della struttura del committente.

Tutti questi dispositivi possono accedere alla rete internet per mezzo di un firewall ed un router, al fine di garantire una maggiore sicurezza dei sistemi interni.

Rispetto all'architettura pre-esistente, viene considerato necessario l'acquisto di uno spazio su un Database Server online. Una possibile soluzione, analizzata all'interno della corrente documentazione, è quella che prevede l'utilizzo dei servizi MySQL.

## 1.2 Architettura software

Di seguito sono riportati, in figura 3.2 ed in figura 3.3, rispettivamente, il deployment diagram ed il component diagram della soluzione proposta per soddisfare la richiesta del committente.

In particolare, per quanto riguarda il deployment diagram (figura 3.2) sono state rappresentate solamente due tipologie di client:

- Client "Dipendenti": è il più *limitato* e permette ai dipendenti solamente la prenotazione per un certo numero di ore di straordinario.
- Client "Uffici": rappresenta sia i computer degli impiegati dell'ufficio amministrazione, che quelli dell'ufficio di gestione del personale. A livello funzionale, questa tipologia di client è considerabile coincidente con il server.

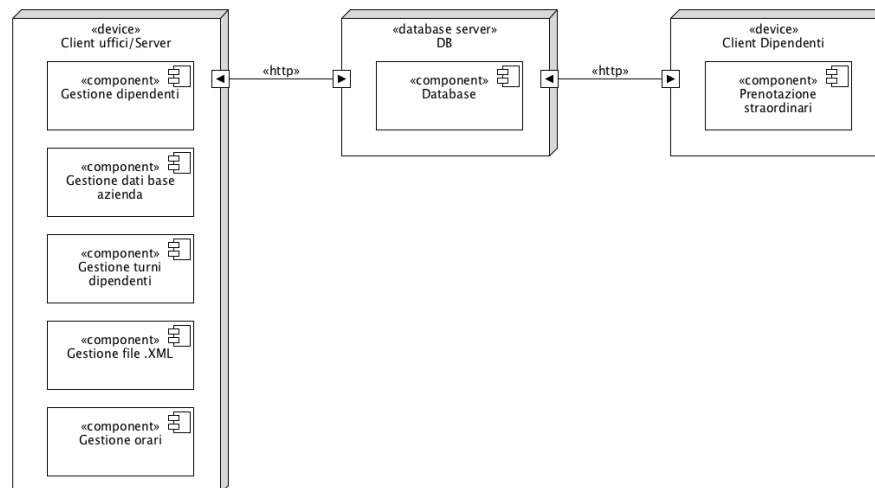


Figura 3.2: Deployment diagram

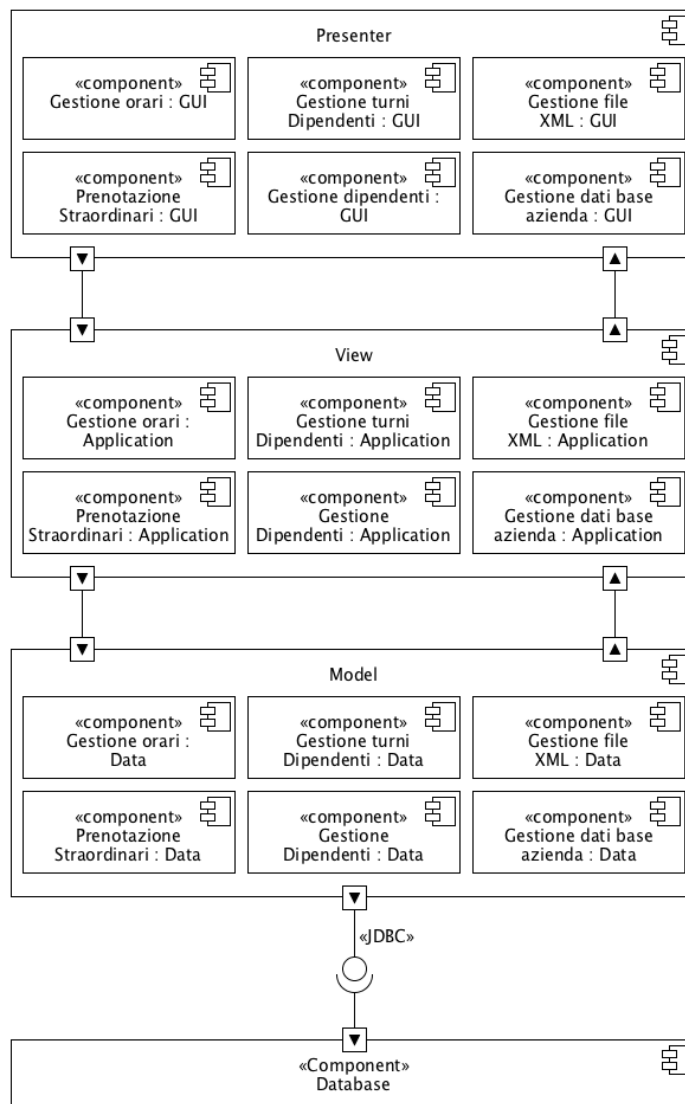


Figura 3.3: Component diagram

Come si nota dalla figura 3.3, ogni singolo componente viene suddiviso in tre blocchi:

- GUI: si occupa di svolgere il ruolo del presenter, ovvero dell'interfaccia grafica.
- Application: ha il compito di implementare la logica di funzionamento di ogni singolo componente.
- Data: permette l'interfaccia con il database.

La modalità di comunicazione tra i vari componenti, tramite metodi offerti dalle interfacce, verrà spiegata nel dettaglio di ogni singolo macro-blocco, nelle pagine seguenti.

### 1.3 Analisi dei componenti

Come evidente dalla figura 3.3, l'intero sistema e tutti i suoi componenti (tranne il database) verranno sviluppati utilizzando il pattern Model-View-Presenter (MVP). Nel MVP, a differenza di MVC, le componenti View e Model interagiscono solo tramite il componente presenter. In questo modo, la View ha solamente il compito di raccogliere gli input dell'utente.

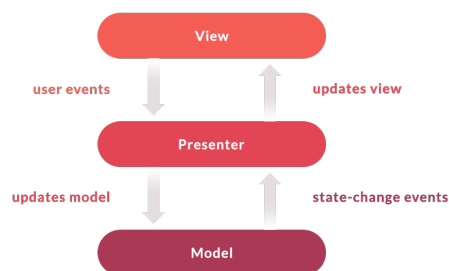


Figura 3.4: Pattern Model-View-Presenter

Questa soluzione permette di personalizzare la modalità di visualizzazione dei contenuti (in base alle eventuali preferenze dell'utente o in base alla categoria di utente che effettua un accesso), senza fare alcuna modifica sui livelli sottostanti.



### 1.3.1 Component Database

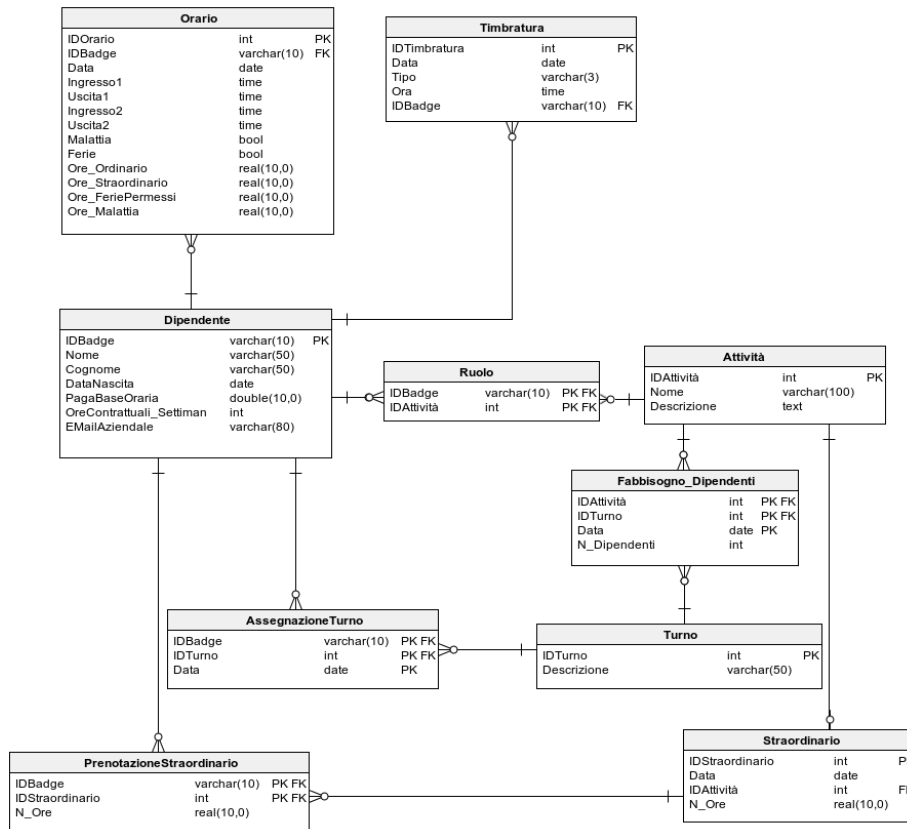


Figura 3.5: Modello del database

Nella figura 3.5 è descritta la struttura del component database che si prevede di utilizzare. In particolare le entità riportate hanno i seguenti significati:

- Dipendente: contiene tutte le informazioni legate al dipendente, sia a livello personale che a livello lavorativo.
- Timbratura: contiene tutte le informazioni ottenute dal parsing del documento .XML prodotto dalla timbratrice.
- Orario: contiene le informazioni elaborate ed aggregate, sulla base delle timbrature fatte dal dipendente.
- Attività: contiene tutte le informazioni riguardanti le attività previste nel processo produttivo dell'azienda committente.
- Ruolo: rappresenta l'implementazione di un "attributo multiplo", ovvero permette di gestire tutte le attività svolte da un dipendente.
- Turno: contiene le principali informazioni di ciascun turno previsto dall'organizzazione del committente.

- Fabbisogno\_Dipendenti: permette di programmare il fabbisogno di dipendenti, per ogni data e per ogni turno, in base alla loro attività ricoperta.
- AssegnazioneTurno: permette di definire l'allocazione dei dipendenti sui vari turni.
- Straordinario: contiene le informazioni, inserite dal committente, utili a gestire la necessità di straordinari, in base all'attività ed alla data.
- PrenotazioneStraordinario: contiene le informazioni riguardanti la "prenotazioni" fatte da ciascun dipendente per una specifica richiesta di straordinari.

### 1.3.2 Component Gestione orari

Il componente di gestione degli orari è composto, come da pattern MVP, da tre macro-componenti:

- **Componente GUI**: Ha il compito di implementare le funzionalità offerte dalla view, ovvero proporre all'utilizzatore un'interfaccia grafica per visualizzare ed, eventualmente, modificare i dati relativi agli orari dei dipendenti.
- **Componente Application**: Ha il compito di implementare la logica di funzionamento della gestione degli orari. Esso, quindi, riceve le informazioni inserite dall'utente dal livello GUI e le elabora combinandole con quelle ottenute dal livello Data.
- **Componente Data**: Ha il compito di offrire un'interfaccia verso il database per il livello Application, ovvero di fornire a tale livello i dati necessari.

Il componente di gestione degli orari necessita di avere un'interfaccia **JDBC/http** per poter accedere ai dati del database che, come riportato all'interno della sottosezione 1.1 del capitolo corrente, si trova su di un server esterno.

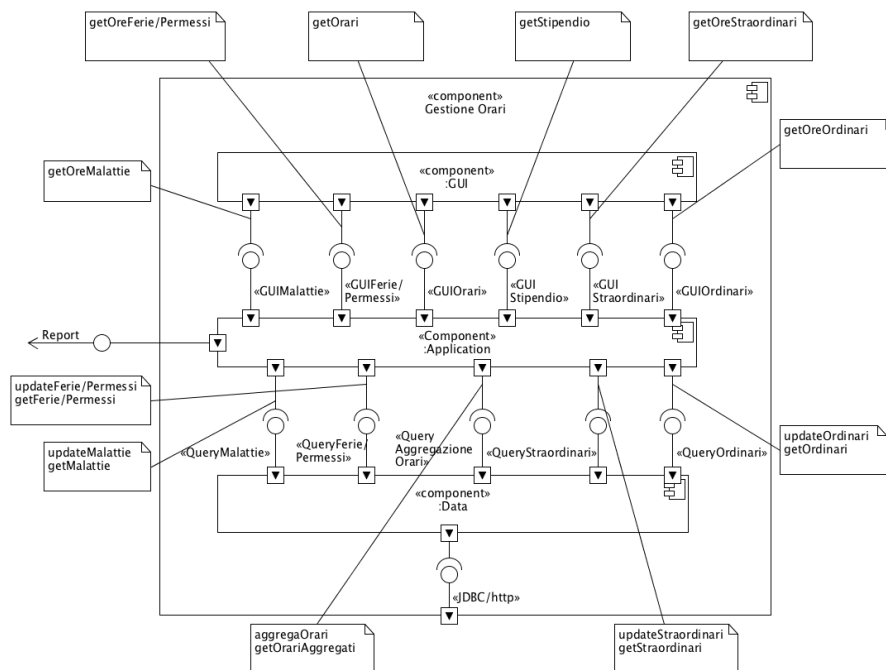
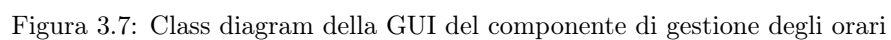


Figura 3.6: Component diagram del componente di gestione degli orari

**GUI** Di seguito, in figura 3.7, è rappresentato il class diagram del componente GUI della gestione degli orari.

Questo componente è pensato in modo da avere una *GestioneOrariView* (che estende la classe *JFrame* delle librerie SWING di Java) nella quale è contenuto un *JPanel*, compilato con un oggetto di una delle altre classi (che estendono la classe *JPanel* delle librerie SWING di Java) in base alla scelta fatta dal menù.



**Application** Il component application sfrutta le interfacce messe a disposizione dal component data (`QueryOrdinari`, `QueryStraordinari`, `QueryAggregazioneOrari`, `QueryPermessi/Ferie` e `QueryMalattie`) per generare, visualizzare e gestire gli orari.

Dato il pattern MVP scelto, il component application espone anche delle interfacce utili per l'interazione della GUI, ovvero `GUIOrdinari`, `GUIStraordinari`, `GUIMalattie`, `GUIOrari`, `GUIFerie/Permessi`, `GUIStipendio` e `Report`.

L'ultima interfaccia esposta dal component application è quella relativa ai report, che possono essere richiesti dall'utente tramite la GUI e che contengono le informazioni riguardanti gli orari inseriti all'interno del database.

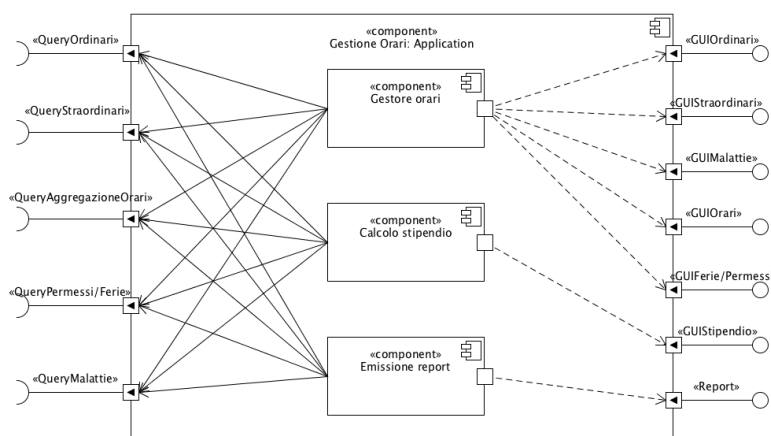


Figura 3.8: Component diagram del componente Application

**Data** Il component data si occupa di recuperare i dati dal database. Il sistema sviluppato è pensato in modo da permettere la persistenza tramite JPA (*Java Persistence API*). In figura 3.9 è mostrato il diagramma delle classi utilizzate per mantenere la persistenza dei dati del database all'interno del sistema software.

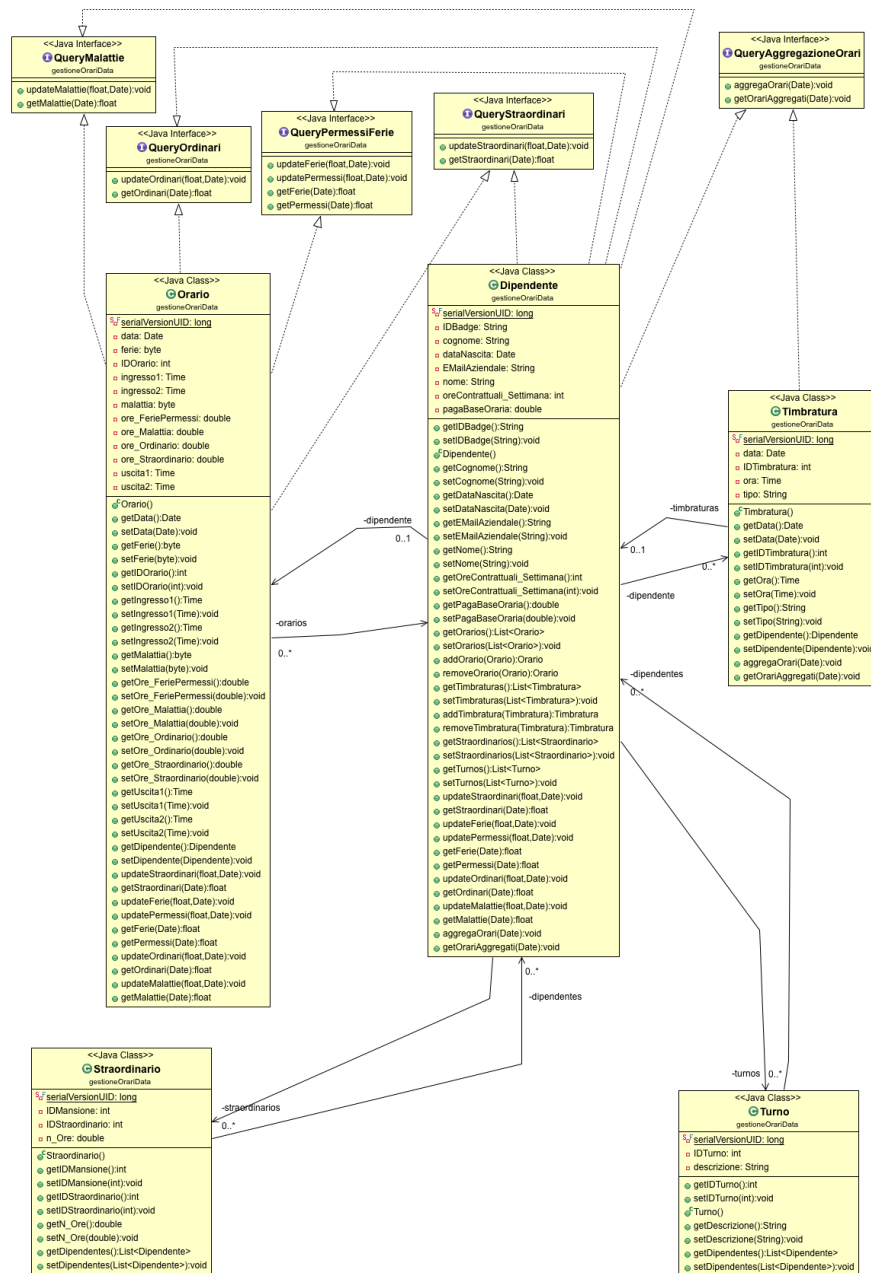


Figura 3.9: Class diagram del componente data del componente di gestione degli orari

### 1.3.3 Component Gestione turni

Il componente di gestione dei turni è composto, come da pattern MVP, da tre macro-componenti:

- Componente **GUI**: Ha il compito di implementare le funzionalità offerte dalla view, ovvero proporre all'utilizzatore un'interfaccia grafica per visualizzare ed, eventualmente, modificare i dati relativi ai turni dei dipendenti.
- Componente **Application**: Ha il compito di implementare la logica di funzionamento della gestione dei turni (ad esempio l'algoritmo di generazione della turnazione settimanale). Esso, quindi, riceve le informazioni inserite dall'utente dal livello **GUI** e le elabora combinandole con quelle ottenute dal livello **Data**.
- Componente **Data**: Ha il compito di offrire un'interfaccia verso il database per il livello **Application**, ovvero di fornire a tale livello i dati necessari.

Il componente di gestione dei turni necessita di avere un'interfaccia **JDBC/http** per poter accedere ai dati del database che, come riportato all'interno della sottosezione 1.1 del capitolo corrente, si trova su di un server esterno.

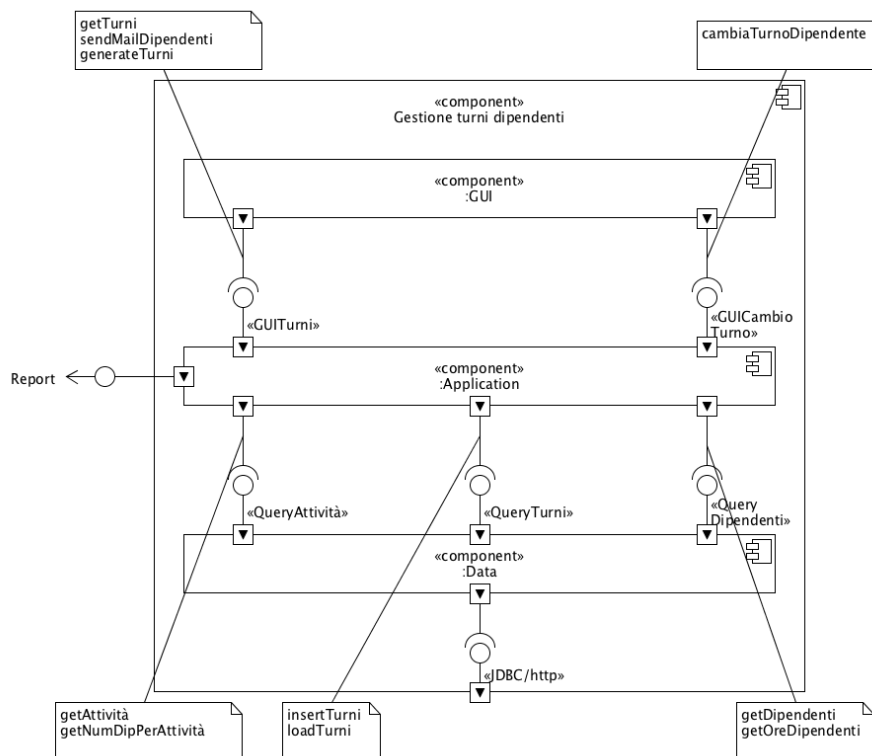


Figura 3.10: Component diagram del componente di gestione dei turni

**GUI** Di seguito, in figura 3.11, è rappresentato il class diagram del componente GUI della gestione dei turni.

Questo componente è pensato in modo da avere una *GestioneTurniView* (che estende la classe *JFrame* delle librerie SWING di Java) nella quale è contenuto

un *JPanel*, compilato con un oggetto di una delle altre classi (che estendono la classe *JPanel* delle librerie SWING di Java) in base alla scelta fatta dal menù.

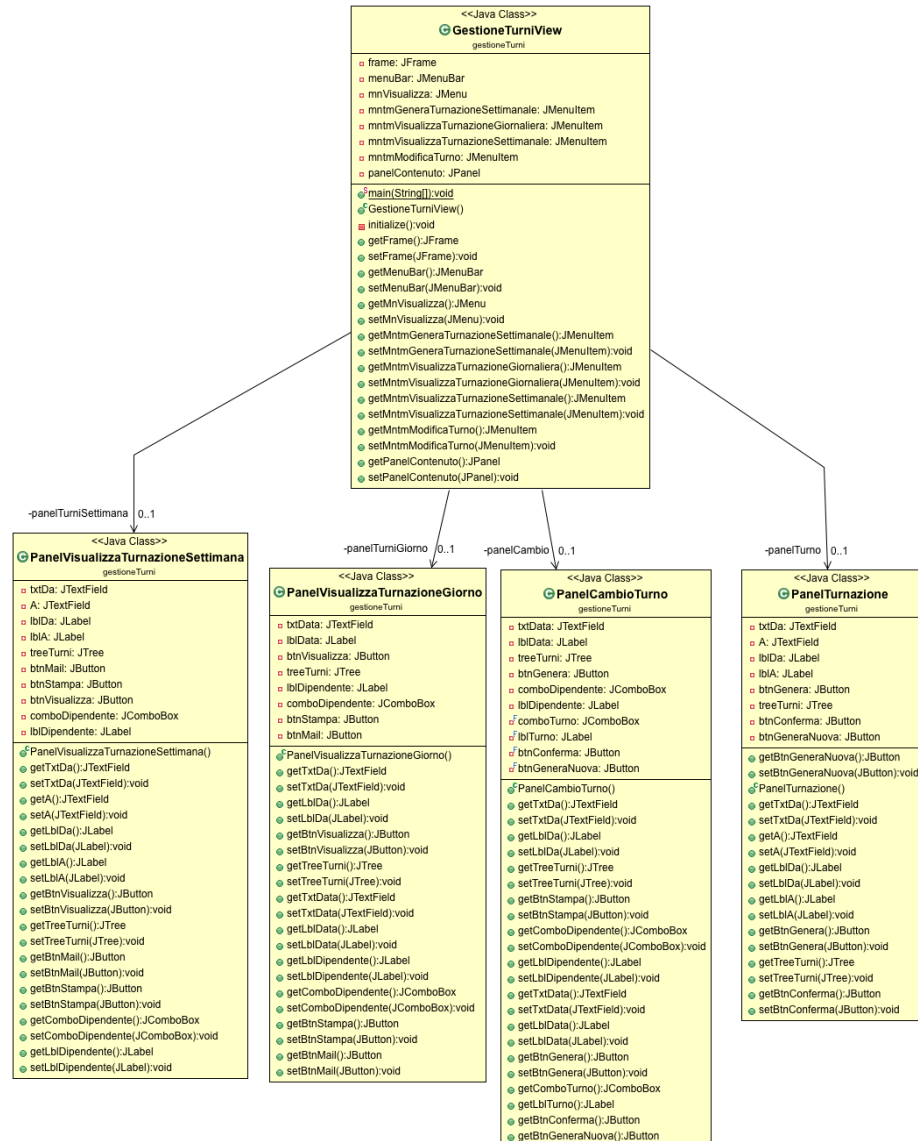


Figura 3.11: Class diagram della GUI del componente di gestione dei turni

**Application** Il component application sfrutta le interfacce messe a disposizione dal component data (*QueryAttività*, *QueryTurni* e *QueryDipendente*) per generare, visualizzare e gestire le turnazioni.



Dato il pattern MVP scelto, il component application espone anche delle interfacce utili per l'interazione della GUI, ovvero `GUITurni` e `GUICambioTurno`.

L'ultima interfaccia esposta dal component application è quella relativa ai report, che possono essere richiesti dall'utente tramite la GUI e che contengono le informazioni riguardanti le turnazioni già generate dal software.

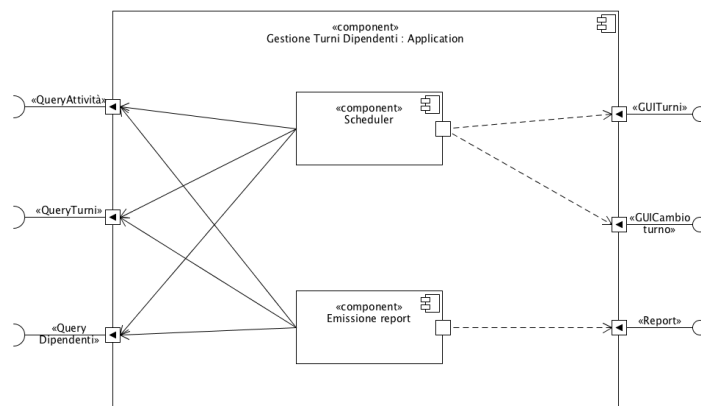


Figura 3.12: Component diagram del componente Application

**Data** Il component data si occupa di recuperare i dati dal database. Il sistema sviluppato è pensato in modo da permettere la persistenza tramite **JPA** (*Java Persistence API*). In figura 3.13 è mostrato il diagramma delle classi utilizzate per mantenere la persistenza dei dati del database all'interno del sistema software.

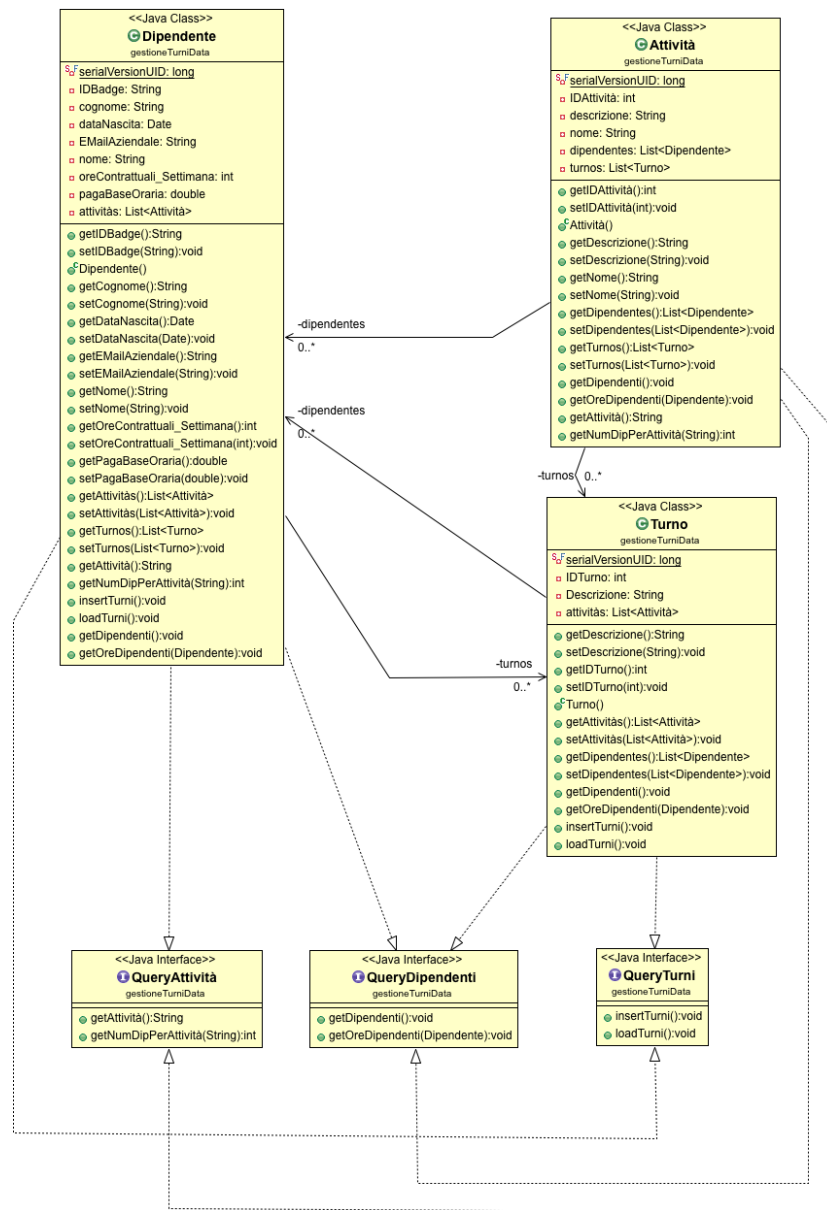


Figura 3.13: Class diagram del componente data del componente di gestione dei turni

Parte II

Fase 2



# Capitolo 4

## Introduzione

### 1 Scelta della funzione da implementare

**Selezione del caso d'uso da implementare** Per questa *Fase 2* del processo di sviluppo dell'applicazione si è scelto di implementare una delle due funzionalità chiave del programma, ovvero la funzionalità descritta all'interno del caso d'uso UC12: Gestione dei dati provenienti dalla timbratrice.

**Formato file di scambio dati** Come accennato anche all'interno della descrizione del caso d'uso, i dati tra la timbratrice ed il software di gestione delle timbrature vengono scambiati tramite file di tipo XML.

**Breve descrizione dell'obiettivo** Si vuole creare una procedura che permetta la selezione, da parte dell'utente, di un file XML prodotto dalla timbratrice e contenente i dati delle timbrature dei dipendenti. L'utente deve inoltre avere la possibilità di aggiungere, modificare o eliminare alcuni dei dati e, successivamente, confermare le modifiche effettuando l'aggregazione dei dati.

**Dati di input** Gli unici dati di input sono contenuti all'interno del file XML:

- Data timbratura
- Tipo di timbratura (IN o OUT)
- Identificativo del badge
- Orario di timbratura

### 2 Passi principali della funzionalità

I passi principali dei quali si compone la funzionalità da implementare saranno:

- Scelta del file XML
- Conversione del file XML in strutture dati (*parsing*)
- Visualizzazione dei dati parsati, con relativa possibilità di:

- Cancellazione dato
  - Modifica dato
- Salvataggio dei dati modificati manualmente
- Aggregazione degli orari e salvataggio definitivo

## Capitolo 5

# Creazione del Database

### 1 Creazione del DB

Per il funzionamento del software è necessario avere su di un Database Server (nella fase di sviluppo sarà `localhost`) il database contenente le tabelle per la memorizzazione delle informazioni.

La soluzione proposta prevede l'utilizzo del DMBS MySQL e le query utilizzate per la generazione del database sono le seguenti:

```
1 CREATE DATABASE Turni ;
2
3 USE Turni ;
4
5 CREATE TABLE AssegnazioneTurno (
6     IDBadge varchar(10) NOT NULL,
7     IDTurno int NOT NULL,
8     Data date NOT NULL,
9     CONSTRAINT AssegnazioneTurno_pk PRIMARY KEY (IDBadge ,
10    IDTurno ,Data)
11 );
12
13 CREATE TABLE Dipendente (
14     IDBadge varchar(10) NOT NULL,
15     Nome varchar(50) NOT NULL,
16     Cognome varchar(50) NOT NULL,
17     DataNascita date NOT NULL,
18     PagaBaseOraria double(10,0) NOT NULL,
19     OreContrattuali_Settimana int NOT NULL,
20     EMailAziendale varchar(80) NOT NULL,
21     CONSTRAINT Dipendente_pk PRIMARY KEY (IDBadge)
22 );
23
24 CREATE TABLE Fabbisogno_Dipendenti (
25     IDAttivita int NOT NULL,
26     IDTurno int NOT NULL,
27     Data date NOT NULL,
28     N_Dipendenti int NOT NULL,
29     CONSTRAINT Fabbisogno_Dipendenti_pk PRIMARY KEY (IDAttivita
30     ,IDTurno ,Data)
31 );
32
33 CREATE TABLE Attivita (
```

```

32     IDAttivita int NOT NULL AUTO_INCREMENT,
33     Nome varchar(100) NOT NULL,
34     Descrizione text NOT NULL,
35     CONSTRAINT Attivita_pk PRIMARY KEY (IDAttivita)
36 );
37
38 CREATE TABLE Orario (
39     IDOrario int NOT NULL AUTO_INCREMENT,
40     IDBadge varchar(10) NOT NULL,
41     Data date NOT NULL,
42     Ingresso1 time,
43     Uscita1 time,
44     Ingresso2 time,
45     Uscita2 time,
46     Malattia bool,
47     Ferie bool,
48     Ore_Ordinario real(10,0),
49     Ore_Straordinario real(10,0),
50     Ore_FeriePermessi real(10,0),
51     Ore_Malattia real(10,0),
52     CONSTRAINT Orario_pk PRIMARY KEY (IDOrario)
53 );
54
55 CREATE TABLE PrenotazioneStraordinario (
56     IDBadge varchar(10) NOT NULL,
57     IDStraordinario int NOT NULL,
58     N_Ore real(10,0) NOT NULL,
59     CONSTRAINT PrenotazioneStraordinario_pk PRIMARY KEY (
60         IDBadge, IDStraordinario)
61 );
62
63 CREATE TABLE Ruolo (
64     IDBadge varchar(10) NOT NULL,
65     IDAttivita int NOT NULL,
66     CONSTRAINT Ruolo_pk PRIMARY KEY (IDBadge, IDAttivita)
67 );
68
69 CREATE TABLE Straordinario (
70     IDStraordinario int NOT NULL AUTO_INCREMENT,
71     Data date NOT NULL,
72     IDAttivita int NOT NULL,
73     N_Ore real(10,0) NOT NULL,
74     UNIQUE INDEX UniqueStraordinarioAttivita (Data, IDAttivita)
75     ,
76     CONSTRAINT Straordinario_pk PRIMARY KEY (IDStraordinario)
77 );
78
79 CREATE TABLE Timbratura (
80     IDTimbratura int NOT NULL AUTO_INCREMENT,
81     Data date NOT NULL,
82     Tipo varchar(3) NOT NULL,
83     Ora time NOT NULL,
84     IDBadge varchar(10) NOT NULL,
85     CONSTRAINT Timbratura_pk PRIMARY KEY (IDTimbratura)
86 );
87
88 CREATE TABLE Turno (
89     IDTurno int NOT NULL AUTO_INCREMENT,
90     Descrizione varchar(50) NOT NULL,
91     CONSTRAINT Turno_pk PRIMARY KEY (IDTurno)
92 );

```



```
92
93 ALTER TABLE AssegnazioneTurno ADD CONSTRAINT
    AssegnazioneTurno_Dipendente FOREIGN KEY
    AssegnazioneTurno_Dipendente (IDBadge)
94 REFERENCES Dipendente (IDBadge);
95
96 ALTER TABLE Orario ADD CONSTRAINT Dipendente_Timbratura FOREIGN
    KEY Dipendente_Timbratura (IDBadge)
97 REFERENCES Dipendente (IDBadge);
98
99 ALTER TABLE Fabbisogno_Dipendenti ADD CONSTRAINT
    Fabbisogno_Dipendenti_Activita FOREIGN KEY
    Fabbisogno_Dipendenti_Activita (IDAttivita)
100 REFERENCES Attivita (IDAttivita);
101
102 ALTER TABLE Fabbisogno_Dipendenti ADD CONSTRAINT
    Fabbisogno_Dipendenti_Turno FOREIGN KEY
    Fabbisogno_Dipendenti_Turno (IDTurno)
103 REFERENCES Turno (IDTurno);
104
105 ALTER TABLE Ruolo ADD CONSTRAINT Attivita_Ruolo FOREIGN KEY
    Attivita_Ruolo (IDAttivita)
106 REFERENCES Attivita (IDAttivita);
107
108 ALTER TABLE Straordinario ADD CONSTRAINT Attivita_Straordinario
    FOREIGN KEY Attivita_Straordinario (IDAttivita)
109 REFERENCES Attivita (IDAttivita);
110
111 ALTER TABLE PrenotazioneStraordinario ADD CONSTRAINT
    PrenotazioneStraordinario_Dipendente FOREIGN KEY
    PrenotazioneStraordinario_Dipendente (IDBadge)
112 REFERENCES Dipendente (IDBadge);
113
114 ALTER TABLE PrenotazioneStraordinario ADD CONSTRAINT
    PrenotazioneStraordinario_Straordinario FOREIGN KEY
    PrenotazioneStraordinario_Straordinario (IDStraordinario)
115 REFERENCES Straordinario (IDStraordinario);
116
117 ALTER TABLE Ruolo ADD CONSTRAINT Ruolo_Dipendente FOREIGN KEY
    Ruolo_Dipendente (IDBadge)
118 REFERENCES Dipendente (IDBadge);
119
120 ALTER TABLE Timbratura ADD CONSTRAINT Timbratura_Dipendente
    FOREIGN KEY Timbratura_Dipendente (IDBadge)
121 REFERENCES Dipendente (IDBadge);
122
123 ALTER TABLE AssegnazioneTurno ADD CONSTRAINT
    Turno_AssignazioneTurno FOREIGN KEY Turno_AssignazioneTurno
    (IDTurno)
124 REFERENCES Turno (IDTurno);
```

Codice 5.1: Query di creazione del database



## Capitolo 6

# Parser XML ed inserimento dati grezzi in DB

### 1 Formato file XML

Per avere un formato del file XML standard, si è deciso di creare un file di descrizione XSD, di seguito riportato.

```
1 <xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org
  /2001/XMLSchema">
2 <xs:element name="records">
3 <xs:complexType>
4 <xs:sequence>
5 <xs:element name="record" maxOccurs="unbounded"
  minOccurs="0">
6 <xs:complexType>
7 <xs:sequence>
8 <xs:element type="xs:byte" name="badgeID"/>
9 <xs:element type="xs:time" name="time"/>
10 </xs:sequence>
11 <xs:attribute type="xs:string" name="type" use="
  optional"/>
12 </xs:complexType>
13 </xs:element>
14 </xs:sequence>
15 <xs:attribute type="xs:string" name="date"/>
16 </xs:complexType>
17 </xs:element>
18 </xs:schema>
```

Codice 6.1: Descrizione XSD del contenuto del file XML

Un esempio di file XML prodotto dalla timbratrice, che verrà utilizzato nelle prove effettuate di seguito, è il seguente:

```
1 <?xml version="1.0" encoding="ISO-8859-1" ?>
2 <records date="30/04/2018">
3 <record type = "IN">
4 <badgeID>00001</badgeID>
5 <time>00:00:00</time>
6 </record>
```

```

7  <record type = "IN">
8    <badgeID>00002</badgeID>
9    <time>07:48:00</time>
10 </record>
11 <record type = "IN">
12   <badgeID>00003</badgeID>
13   <time>16:01:00</time>
14 </record>
15 <record type = "OUT">
16   <badgeID>00001</badgeID>
17   <time>08:02:00</time>
18 </record>
19 <record type = "OUT">
20   <badgeID>00002</badgeID>
21   <time>16:00:00</time>
22 </record>
23 <record type = "OUT">
24   <badgeID>00003</badgeID>
25   <time>23:59:00</time>
26 </record>
27 </records>

```

Codice 6.2: Esempio di file XML prodotto dalla timbratrice

## 2 Set-up preventivo

Nella fase di importazione delle timbrature, gli unici dati che devono essere a disposizione del sistema sono quelli relativi ai dipendenti (*in particolare quelli legati ai badgeID*). In questa prima fase dello sviluppo del software, però, non è ancora presente la componente di gestione dei dati dei dipendenti.

La soluzione scelta per riuscire comunque ad implementare, testare ed utilizzare la funzionalità è quella di effettuare in inserimento di dati di prova all'interno del database grazie ad una semplice query SQL:

```

1 INSERT INTO Dipendente (IDBadge, Nome, Cognome, DataNascita,
   PagaBaseOraria, OreContrattuali_Settimana, EMailAziendale)
VALUES ("00001", "Paolo", "Rossi", "2010-10-10", 8,40, "paolo.
rossi@prova.it"), ("00002", "Marco", "Bianchi", "2001-01-01",
8,40, "marco.bianchi@prova.it"), ("00003", "Bruno", "Neri", "
2002-02-02", 8,40, "bruno.neri@prova.it");

```

Codice 6.3: Query SQL per l'inserimento dei dati preventivi

## 3 Implementazione del parser

Il file XML viene convertito in una `List<Timbratura>` utilizzando la classe `SAXParser` già presente nella libreria standard di Java:

```

1 public interface XMLParser<T> {
2   public T parserXML(String filePath);
3 }
4
5 public class RecordParser implements
6   XMLParser<List<Timbratura>> {

```

```
7
8  @Override
9  public List<Timbratura> parserXML(String filePath) {
10     try {
11         SAXParserFactory factory =
12             SAXParserFactory.newInstance();
13         SAXParser saxParser = factory.newSAXParser();
14         XMLHandler recordHandler = new XMLHandler();
15         saxParser.parse(new InputSource(
16             new FileReader(filePath)), recordHandler);
17         return recordHandler.getRecordList();
18     }
19     catch (Exception e) {
20         e.printStackTrace();
21         return null;
22     }
23 }
24 }
```

Codice 6.4: Classe che si occupa di effettuare il parsing del file XML

Il metodo `parserXML` della classe `RecordParser` permette di ottenere una `List<Timbratura>` che può, quindi, essere inserita all'interno del database nella relativa tabella.

In questa classe viene utilizzato anche un oggetto della classe `XMLHandler`, implementata estendendo la classe `DefaultHandler` che permette la gestione automatica del processo di lettura di un file XML.

## 4 Funzionamento della classe XMLHandler

La classe `XMLHandler` permette la gestione automatica del processo di lettura di un file XML con il seguente funzionamento:

- Quando si incontra l'apertura di un nuovo tag, viene chiamato in automatico il metodo `startElement`.  
Due importanti parametri di questo metodo sono `qName`, che contiene il nome del tag incontrato, e `attributes`, che contiene l'elenco di tutti gli eventuali attributi del tag.
- Se all'interno del tag sono presenti dei dati, viene chiamato il metodo `characters`.
- Quando si incontra la terminazione del tag, viene chiamato in automatico il metodo `endElement`.

```
1 public class XMLHandler extends DefaultHandler{
2
3     private XMLTag tag = XMLTag.NOTAG;
4     private Date dataXML;
5     private List<Timbratura> listaTimbrature = new ArrayList<
6         Timbratura>();
7     private Timbratura tempRecord = new Timbratura();
8
9     public void startElement(String uri, String localName, String
10         qName, Attributes attributes) throws SAXException {
11         ...
12     }
```

```
10 }
11
12 public void characters(char[] ch, int start, int length)
13     throws SAXException {
14     ...
15 }
16
17 public void endElement(String uri, String localName, String
18     qName) throws SAXException {
19     ...
20 }
21
22 public List<Timbratura> getRecordList() {
23     ...
24 }
```

Codice 6.5: Classe XMLHandler

#### 4.1 Identificazione dei tag

La lista di tutti i tag possibili del file XML è contenuta nella `enum XMLTag` e viene utilizzata da tutti i metodi della classe `XMLHandler` per decidere quale azione deve essere svolta, tag per tag.

```
1 public enum XMLTag {
2     NOTAG, RECORDS, RECORD, BADGEID, TIME;
3 }
```

Codice 6.6: Enumerazione XMLTag

Sulla base del tag incontrato, la funzione `startElement` è in grado di capire se:

- Ci si trova all'inizio del file
- Si è nella necessità di creare una nuova `Timbratura`
- Devono essere inseriti dei dati in una `Timbratura` creata in precedenza

```
1 public void startElement(String uri, String localName, String
2     qName, Attributes attributes) throws SAXException {
3     switch (qName) {
4         case "records":
5             tag = XMLTag.RECORDS;
6             dataXML = new Date(Integer.parseInt(attributes.getValue("
7                 date").toString().substring(6, 10))-1900,
8                 Integer.parseInt(attributes.getValue("date").toString
9                 ().substring(3, 5))-1,
10                 Integer.parseInt(attributes.getValue("date").toString
11                 ().substring(0, 2)));
12             break;
13         case "record": tag = XMLTag.RECORD;
14             tempRecord = new Timbratura();
15             tempRecord.setData(dataXML);
16             tempRecord.setTipo(attributes.getValue("type"));
17             break;
18         case "badgeID": tag = XMLTag.BADGEID; break;
19         case "time": tag = XMLTag.TIME; break;
```

```
16     default: break;
17   }
18 }
```

Codice 6.7: Metodo startElement

## 4.2 Selezione del contenuto del tag

Noto il tag in cui ci si trova durante il processo di parsing, i dati interni possono essere salvati all'interno dell'oggetto di classe **Timbratura**.

```
1 public void characters(char[] ch, int start, int length) throws
   SAXException {
2   String content = new String(ch, start, length);
3   switch (tag) {
4     case BADGEID:
5       TypedQuery<Dipendente> query = DBHandler.getEntityManager
6         ().createQuery("SELECT d FROM Dipendente d WHERE d.IDBadge
7           ='" + content + "'",Dipendente.class);
8       if (query.getResultList().isEmpty() || query.
9         getResultList().size()>1) throw new SAXException();
10      tempRecord.setDipendente(query.getResultList().get(0));
11      break;
12     case TIME:
13       tempRecord.setOra(new Time(Integer.parseInt(content.
14         substring(0, 2)),
15         Integer.parseInt(content.substring(3,5)),
16         Integer.parseInt(content.substring(6,8))));
17       break;
18     default: break;
19   }
20 }
```

Codice 6.8: Metodo characters

## 4.3 Chiusura del tag

E' importante non considerare solamente l'apertura di un tag ma anche la sua chiusura poichè, nel file XML prodotto dalla timbratrice si hanno tag interni alla **Timbratura**. Per questo motivo ci sono dei tag che devono dare origine alla creazione di un oggetto di classe **Timbratura** e tag che invece sono relativi solamente ad alcuni campi di una **Timbratura**.

La decisione sul tipo di tag incontrato viene fatta, come accennato in precedenza, dal metodo **endElement**.

```
1 public void endElement(String uri, String localName, String
   qName) throws SAXException {
2   switch (tag) {
3     case TIME:
4       listaTimbrature.add(tempRecord);
5     default:
6       tag=XMLTag.NOTAG;
7   }
8 }
```

Codice 6.9: Metodo endElement

## 4.4 Risultato del Parsing

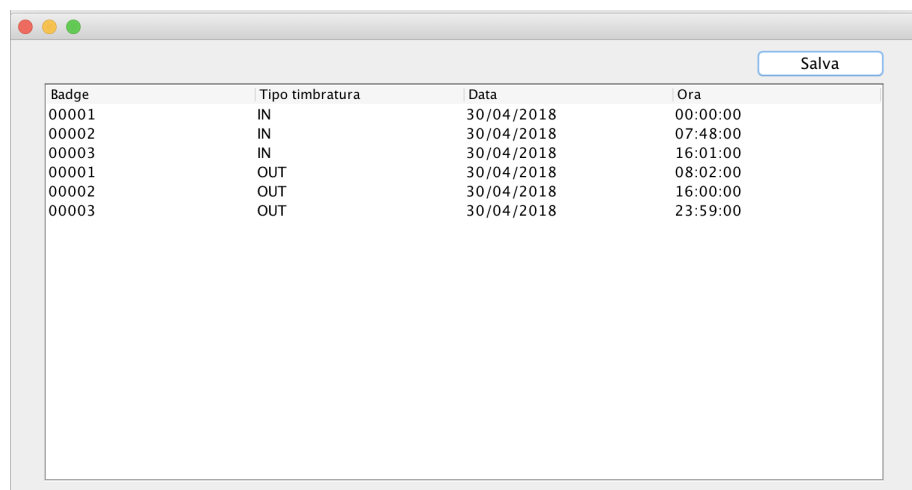
Il risultato dell'operazione di parsing è una `List<Timbratura>`, che può essere ottenuta tramite la chiamata del metodo `getRecordList()`.

```
1 public List<Timbratura> getRecordList () {  
2     return listaTimbrature;  
3 }
```

Codice 6.10: Metodo `getRecordList`

## 5 Inserimento dei dati grezzi nel database

Una volta che il file XML è stato correttamente parsato, i dati vengono mostrati in una maschera dell'applicativo.



| Badge | Tipo timbratura | Data       | Ora      |
|-------|-----------------|------------|----------|
| 00001 | IN              | 30/04/2018 | 00:00:00 |
| 00002 | IN              | 30/04/2018 | 07:48:00 |
| 00003 | IN              | 30/04/2018 | 16:01:00 |
| 00001 | OUT             | 30/04/2018 | 08:02:00 |
| 00002 | OUT             | 30/04/2018 | 16:00:00 |
| 00003 | OUT             | 30/04/2018 | 23:59:00 |

Figura 6.1: Finestra per la modifica degli orari

Come da UC12, con il click sul pulsante **Salva**, i dati grezzi (*con le eventuali correzioni manuali*) devono essere inseriti all'interno del database dell'applicazione.

Questa operazione viene fatta, sfruttando le API JPA tramite il codice seguente:

```
1 Timbratura tempTimb = new Timbratura();  
2 ...  
3 ...  
4 ...  
5 DBHandler.getEntityManager().getTransaction().begin();  
6 DBHandler.getEntityManager().persist(tempTimb);  
7 DBHandler.getEntityManager().getTransaction().commit();
```

Codice 6.11: Salvataggio dei dati grezzi provenienti dalla timbratrice



## Capitolo 7

# Aggregazione degli orari

### 1 Introduzione

Una volta che i dati grezzi, relativi alle singole timbrature, sono stati inseriti all'interno del database, il passo successivo è quello di aggregare i dati di ogni singolo giorno per ogni dipendente.

Di seguito verrà riportata l'analisi di questa funzionalità.

### 2 Funzione di aggregazione

Il compito della funzione di aggregazione può essere riassunto dai seguenti passi:

- Individuare tutte le timbrature fatte da un determinato dipendente in una determinata data.
- Ordinare le timbrature individuate in ordine crescente.
- Creare un "orario" completo, comprendente tutte le entrate ed uscite giornaliere del dipendente.
- Per ogni giornata di lavoro calcolare:
  - Ore di lavoro ordinario
  - Ore di lavoro straordinario
  - Ore di ferie/permesso

La funzione può essere implementata dal seguente pseudocodice:

```
1 funzione aggregaOrari
2   dataMin <- min(data timbrature inserite)
3   dataMax <- max(data timbrature inserite)
4   creaRigheVuoteOrari(dataMax, dataMin)
5   posizionaOrari(dataMax, dataMin)
6   elaboraOrari(dataMax, dataMin)
7   eliminaDatiTimbrature()
8 fine funzione
```

Codice 7.1: Pseudocodice della funzione di aggregazione

Come da Codice 7.1, la funzione **aggregaOrari** comprende al suo interno diverse chiamate a procedure che si occupano di svolgere i vari passi necessari al processo di aggregazione.

## 2.1 Funzione di creazione delle righe vuote

La funzione di creazione delle righe vuote si occupa di predisporre la tabella degli **Orari**, completandola con tutte le giornate, per ogni dipendente, comprese tra la minima e la massima data contenuta all'interno del file XML importato dalla timbratrice.

La funzione può essere rappresentata dal *flow-chart* in figura 7.1 ed implementata dallo pseudocodice riportato in Codice 7.2.

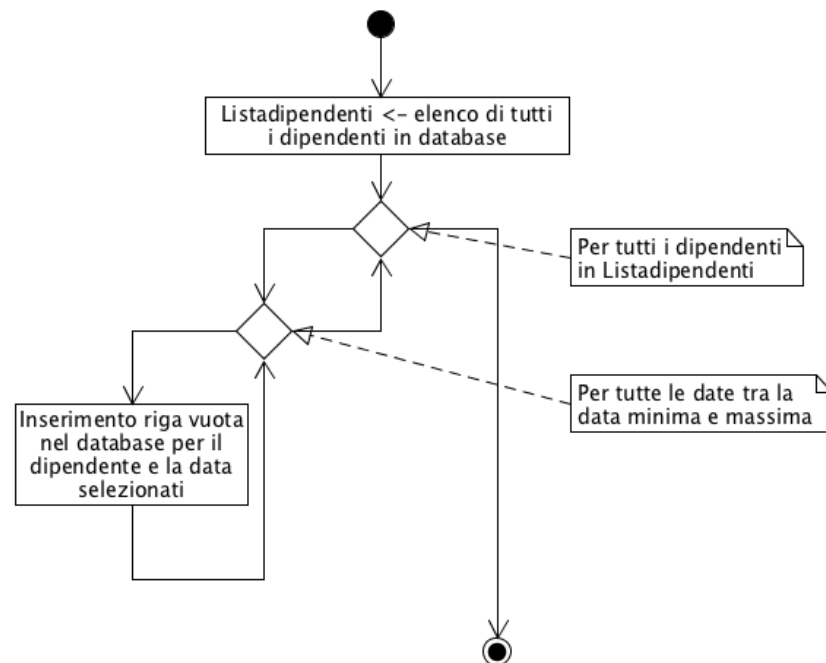


Figura 7.1: Flow-chart per la funzione di creazione delle righe vuote

```

1 funzione creaRigheVuoteOrari(dataMax, dataMin)
2   Listadipendenti <- elenco di tutti i dipendenti in database
3   for each dipendente in Listadipendenti
4     for each data between dataMin and dataMax
5       inserimento di una riga vuota per [dipendente] nella data
6       [data] nella tabella degli orari aggregati
7     end for
8   end for
9 fine funzione
  
```

Codice 7.2: Pseudocodice della funzione di creazione delle righe vuote

### 2.1.1 Analisi della complessità

Consideriamo:

- $n$ , il numero di dipendenti dell'azienda
- $m$ , il numero di giorni tra `dataMin` e `dataMax`

La complessità di questa funzione, considerando le operazioni di accesso al database di costo unitario  $O(1)$  (*semplificazione*), può essere considerata pari a

$$O(n \cdot m) \tag{7.1}$$

Inoltre, supponendo di lavorare con un committente che abbia un numero significativo di dipendenti e che importi i dati della timbratrice con un intervallo abbastanza breve, possiamo considerare

$$n \gg m \tag{7.2}$$

ovvero, passiamo ad una complessità della funzione pari a

$$O(n) \tag{7.3}$$

## 2.2 Funzione di posizionamento degli orari

La funzione di posizionamento degli orari si occupa di copiare, in ciascuna delle righe della tabella degli `Orari` predisposte dalla funzione precedente, tutte le timbrature effettuate da un dipendente in una specifica giornata. In modo è possibile avere la situazione complessiva delle entrate e delle uscite del dipendente dall'area produttiva.

La copia delle timbrature deve essere effettuata in ordine orario crescente.

La funzione può essere rappresentata dal *flow-chart* in figura 7.2 ed implementata dallo pseudocodice riportato in Codice 7.3.

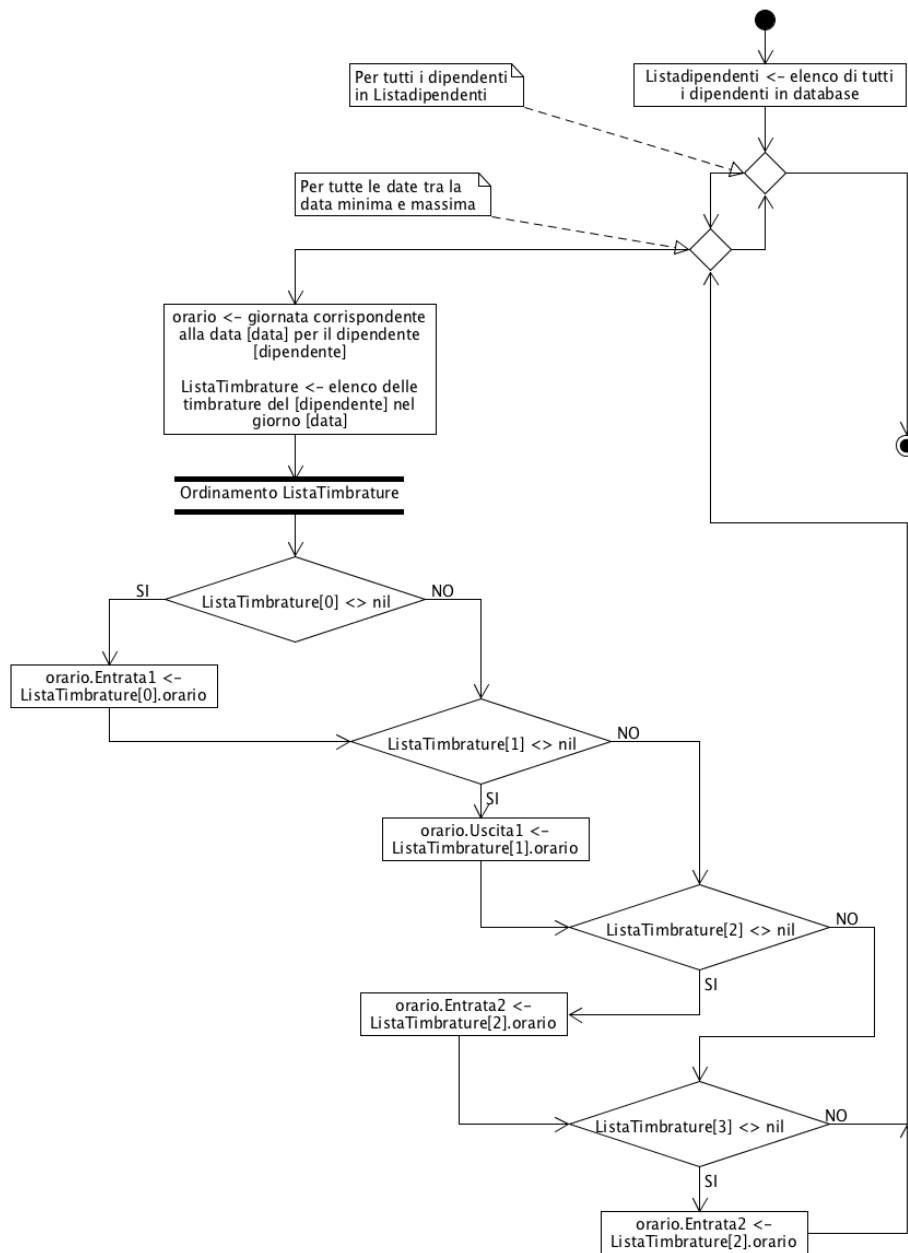


Figura 7.2: Flow-chart per la funzione di posizionamento degli orari

```

1 funzione posizionaOrari(dataMax, dataMin)
2   Listadipendenti <- elenco di tutti i dipendenti in database
3   for each dipendente in Listadipendenti
4     for each data between dataMin and dataMax
5       orario <- giornata corrispondente alla data [data] per
6       il dipendente [dipendente]
7       ListaTimbrature <- elenco delle timbrature del

```

```

8      [dipendente] nel giorno [data]
9      ordina(ListaTimbrature) in ordine di orario
10     if ListaTimbrature[0] <> nil then
11         orario.Entrata1 <- ListaTimbrature[0].orario
12     end if
13     if ListaTimbrature[1] <> nil then
14         orario.Uscita1 <- ListaTimbrature[1].orario
15     end if
16     if ListaTimbrature[2] <> nil then
17         orario.Entrata2 <- ListaTimbrature[2].orario
18     end if
19     if ListaTimbrature[3] <> nil then
20         orario.Uscita2 <- ListaTimbrature[3].orario
21     end if
22     salva dati in database
23 end for
24 end for
25 fine funzione

```

Codice 7.3: Pseudocodice della funzione di posizionamento degli orari

### 2.2.1 Analisi della complessità

Consideriamo:

- $n$ , il numero di dipendenti dell'azienda
- $m$ , il numero di giorni tra `dataMin` e `dataMax`
- Ordinamento eseguito tramite un algoritmo ottimo, con complessità  $O(n \cdot \log n)$ .  
Questa assunzione vale solamente a livello teorico, in quanto l'ordinamento verrà eseguito tramite una query SQL e, di conseguenza, gestito dal DMBS con algoritmo non-noto.

La complessità di questa funzione, considerando le operazioni di accesso al database di costo unitario  $O(1)$  (*semplificazione*), può essere considerata pari a

$$O(n \cdot m \cdot n \cdot \log n) = O(n^2 \cdot m \cdot \log n) \quad (7.4)$$

Inoltre, supponendo di lavorare con un committente che abbia un numero significativo di dipendenti e che importi i dati della timbratrice con un intervallo abbastanza breve, possiamo considerare

$$n \gg m \quad (7.5)$$

ovvero, passiamo ad una complessità della funzione pari a

$$O(n^2 \cdot \log n) \quad (7.6)$$

## 2.3 Funzione di elaborazione degli orari

La funzione di elaborazione degli orari deve, basandosi su quanto prodotto dalla funzione precedente, calcolare, per ogni giornata e per ogni dipendente, le ore di:

- Ordinario

- Straordinario
- Ferie/Permessi

Per scelta del committente, gli il numero di ore viene gestito a scatti di mezz'ora, arrotondando alla mezz'ora più vicina.

La funzione può essere rappresentata dal *flow-chart* in figura 7.3 ed implementata dallo pseudocodice riportato in Codice 7.4.

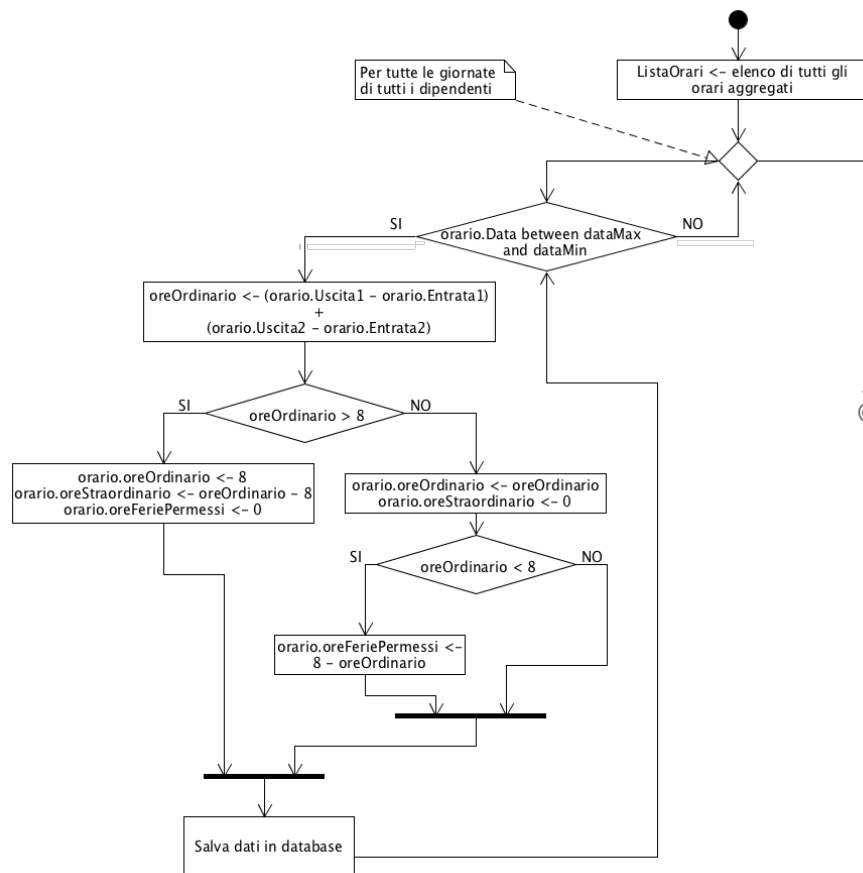


Figura 7.3: Flow-chart per la funzione di elaborazione degli orari

```

1 funzione elaboraOrari(dataMax, dataMin)
2   ListaOrari <- elenco di tutti gli orari aggregati
3   for each orario in ListaOrari
4     if orario.Data between dataMax and dataMin then
5       oreOrdinario <- (orario.Uscita1 - orario.Entrata1) + (
6         orario.Uscita2 - orario.Entrata2)
7       if oreOrdinario > 8 then
8         orario.oreOrdinario <- 8
9         orario.oreStraordinario <- oreOrdinario - 8

```

```
9      orario.oreFeriePermessi <- 0
10    else
11      orario.oreOrdinario <- oreOrdinario
12      orario.oreStraordinario <- 0
13      if oreOrdinario < 8 then
14        orario.oreFeriePermessi <- 8 - oreOrdinario
15      end if
16    end if
17  end if
18  salva dati in database
19 end for
20 fine funzione
```

Codice 7.4: Pseudocodice della funzione di elaborazione degli orari

### 2.3.1 Analisi della complessità

Consideriamo:

- $k$ , il numero di righe inserite nella tabella degli **Orari** aggregati

La complessità di questa funzione, considerando le operazioni di accesso al database di costo unitario  $O(1)$  (*semplificazione*), può essere considerata pari a

$$O(k) \quad (7.7)$$

## 2.4 Funzione di eliminazione dei dati grezzi

Una volta che gli orari sono stati aggregati ed elaborati correttamente, per ottimizzare lo spazio, è richiesta la cancellazione dei dati delle **Timbrature** grezze, importate dal file XML prodotto dalla timbratrice.

Questa operazione, concettualmente molto semplice, viene svolta dallo pseudocodice in Codice 7.5.

```
1 funzione eliminaDatiTimbrature()
2   elimina dal database tutte le timbrature grezze
3 fine funzione
```

Codice 7.5: Pseudocodice della funzione di eliminazione delle timbrature grezze

### 2.4.1 Analisi della complessità

Questa funzione effettua solamente delle operazioni di accesso al database, quindi per le assunzioni fatte in precedenza possiamo considerare la sua complessità pari a

$$O(1) \quad (7.8)$$

## 2.5 Analisi della complessità generale

Guardando l'algoritmo di aggregazione ed elaborazione degli orari nella sua completezza, è possibile stabilire la sua complessità come la somma di tutte le complessità delle singole funzioni che lo compongono.

In particolare, considerando

- $n$ , il numero di dipendenti dell'azienda
- $m$ , il numero di giorni tra `dataMin` e `dataMax`
- $k$ , il numero di giornate inserite nella tabella degli `Orari` aggregati
- Ordinamento eseguito tramite un algoritmo ottimo, con complessità  $O(n \cdot \log n)$ .  
Questa assunzione vale solamente a livello teorico, in quanto l'ordinamento verrà eseguito tramite una query SQL e, di conseguenza, gestito dal DMBS con algoritmo non-noto.

e mantenendo valide tutte le ipotesi semplificative fatte nelle singole analisi, la complessità totale della funzionalità di aggregazione risulta pari a

$$O(n) + O(n^2 \cdot \log n) + O(k) + O(1) = O(n^2 \cdot \log n) \quad (7.9)$$



## Capitolo 8

# Test ed analisi del componente implementato

### 1 Analisi statica

Attraverso il tool **Stan4j** è stata effettuata l'analisi statica della porzione del software sviluppata in questa fase. Analizzando quanto riportato in Figura 8.1 risulta rispettata la struttura MVP che ci si era proposti di realizzare.

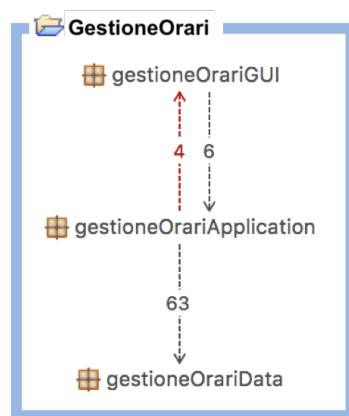


Figura 8.1: Analisi statica del codice

### 2 Analisi dinamica - testing

In questa sezione ci si concentrerà sull'implementazione di un certo numero di casi di test al fine di avere la massima copertura possibile del codice della porzione di software implementato in questa fase.

Data la suddivisione, tramite pattern MVP, del codice in `GUI`, `Application` e `Data` è stato scelto di testare principalmente la componente `Application`, tralasciando i due `Listener`, in quanto:

- La componente `GUI` è difficile da testare
- La componente `Data` è stata generata in automatico grazie alle API JPA.

Inoltre, al fine di mantenere la migliore organizzazione possibile, si è scelto di inserire i casi di test in un package a parte, denominato `gestioneOrariTest`.

## 2.1 Test della classe `DateValidator`

Per testare la classe `DateValidator`, utilizzata nel processo di salvataggio nel database dei dati modificati manualmente da parte dell'impiegato dell'ufficio amministrazione, si è scelto di analizzare, per la funzione di verifica del formato della data (`isThisDateValid`), i seguenti casi:

- Data e formato di controllo validi
- Data errata ma formato di controllo valido:
  1. Giorno non esistente
  2. Mese non esistente
  3. Data non numerica
- Formato di controllo errato
- Controllo su 29 febbraio nel caso di anno bisestile ed anno non bisestile
- Data `null`

Per quanto riguarda la funzione `addDays`, usata nella somma di un certo numero di giorni ad una data, sono stati analizzati i casi seguenti:

- Data corretta
  1. Con numero di giorni positivo
  2. Con numero di giorni negativo
  3. Con numero di giorni pari a zero
- Data `null`

I casi di test sono stati eseguiti tramite `JUnit` ed il codice riportato in Codice 8.1.

```
1 package gestioneOrariTest;
2
3 import static org.junit.jupiter.api.Assertions.*;
4 import java.text.ParseException;
5 import org.junit.jupiter.api.Test;
6 import gestioneOrariApplication.DateValidator;
7
8 class DateValidatorTest {
9
10  @Test
```

```

11 void testDataCorretta() throws ParseException {
12     DateValidator.isThisDateValid("10/04/2018", "dd/MM/yyyy");
13 }
14
15 @Test
16 void testDataGiornoErrato() {
17     assertThrows(ParseException.class, () -> {
18         DateValidator.isThisDateValid("31/04/2018", "dd/MM/yyyy");
19     });
20     assertThrows(ParseException.class, () -> {
21         DateValidator.isThisDateValid("04/14/2018", "dd/MM/yyyy");
22     });
23     assertThrows(ParseException.class, () -> {
24         DateValidator.isThisDateValid("GGDBEJK", "dd/MM/yyyys");
25     });
26 }
27
28 @Test
29 void formatoDataErrato() {
30     assertThrows(IllegalArgumentException.class, () -> {
31         DateValidator.isThisDateValid("31/04/2018", "ss/mm/pqrt");
32     });
33 }
34
35 @Test
36 void annoBisestile() throws ParseException {
37     DateValidator.isThisDateValid("29/02/2020", "dd/MM/yyyy");
38     assertThrows(ParseException.class, () -> {
39         DateValidator.isThisDateValid("20/02/2018", "dd/MM/yyyys");
40     });
41 }
42
43 @Test
44 void dataNull() {
45     assertThrows(ParseException.class, () -> {
46         DateValidator.isThisDateValid(null, "dd/MM/yyyys");
47     });
48 }
49
50 @Test
51 void addDataCorretto() {
52     assertEquals(DateValidator.addDays(new Date(2018,2,3),1).
53         getDate(),4);
54     assertEquals(DateValidator.addDays(new Date(2018,2,3),0).
55         getDate(),3);
56     assertEquals(DateValidator.addDays(new Date(2018,2,3),-1).
57         getDate(),2);
58 }
59
60 @Test
61 void addDataNull() {
62     assertThrows(NullPointerException.class, () -> {
63         DateValidator.addDays(null,1);
64     });
65 }

```

Codice 8.1: Casi JUnit per il test della classe DateValidator

Tutti i casi di test hanno dato esito positivo (Figura 8.2) ed hanno permesso di raggiungere un coverage della classe `DateValidator` dell'91.7% delle istruzioni (figura 8.3). Avere una copertura maggiore dell'91.7% risulta impossibile, in

quanto nel calcolo del numero delle istruzioni viene anche considerata la riga di intestazione della classe e le righe dei commenti.

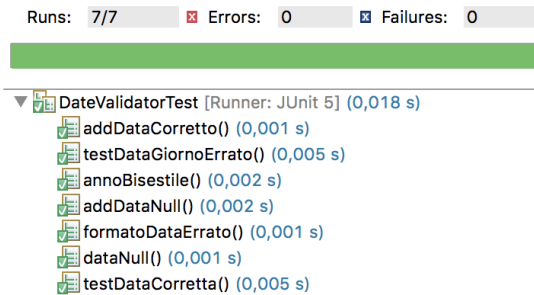


Figura 8.2: Risultato dell'esecuzione dei casi di test

| Coverage                                 |          |                      |                     |                    |
|--|----------|----------------------|---------------------|--------------------|
| DateValidatorTest (10-ago-2018 19.02.14) |          |                      |                     |                    |
| Element                                  | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
| DateValidator.java                       | 91,7 %   | 21                   | 3                   | 24                 |

Figura 8.3: Livello di copertura dei casi di test

## 2.2 Test della classe TimeValidator

Per testare la classe `TimeValidator`, utilizzata nel processo di salvataggio nel database dei dati modificati manualmente da parte dell'impiegato dell'ufficio amministrazione, per la funzione di verifica del formato dell'orario (`validate`), si è scelto di analizzare i seguenti casi:

- Orari validi
- Orari con formato diverso da quello valido (*hh : mm : ss*)
- Orari non esistenti:
  1. Ora superiore a 23
  2. Minuti superiori a 59
  3. Secondi superiori a 59

Per quanto riguarda la funzione `HoursSet`, usata per decrementare di *1h* l'orario, prima di inserirlo nel database MySQL, sono stati analizzati i casi seguenti:

- Orario corretto
- Orario `null`

I casi di test sono stati eseguiti tramite `JUnit` ed il codice riportato in Codice 8.2.

```
1 package gestioneOrariTest;
2
3 import static org.junit.jupiter.api.Assertions.*;
4 import org.junit.jupiter.api.Test;
5 import gestioneOrariApplication.TimeValidator;
6
7 class TimeValidatorTest {
8
9     @Test
10    void testOrarioCorretto() throws Exception {
11        TimeValidator.validate("10:20:00");
12        TimeValidator.validate("23:59:59");
13        TimeValidator.validate("00:00:00");
14    }
15
16    @Test
17    void testOrarioFormatoErrato() {
18        assertThrows(Exception.class, () -> {
19            TimeValidator.validate("10-20-00");
20        });
21        assertThrows(Exception.class, () -> {
22            TimeValidator.validate("10:20.00");
23        });
24        assertThrows(Exception.class, () -> {
25            TimeValidator.validate("10.20.00");
26        });
27    }
28
29    @Test
30    void testOrarioErrato() {
31        assertThrows(Exception.class, () -> {
32            TimeValidator.validate("24:00:00");
33        });
34        assertThrows(Exception.class, () -> {
35            TimeValidator.validate("25:00:00");
36        });
37        assertThrows(Exception.class, () -> {
38            TimeValidator.validate("10:70:00");
39        });
40        assertThrows(Exception.class, () -> {
41            TimeValidator.validate("10:20:80");
42        });
43    }
44
45    @Test
46    void testSetOreCorretto() {
47        Date d = new Date();
48        d.setHours(5);
49        assertEquals(TimeValidator.HoursSet(d).getHours(), 4);
50    }
51
52    @Test
53    void testSetOreNull() {
54        assertThrows(NullPointerException.class, () -> {
55            TimeValidator.HoursSet(null);
56        });
57    }
58 }
```

Codice 8.2: Casi JUnit per il test della classe TimeValidator

Tutti i casi di test hanno dato esito positivo (Figura 8.4) ed hanno permesso di raggiungere un coverage della classe `TimeValidator` del 90.3% delle istruzioni (figura 8.5). Avere una copertura maggiore dell'90.3% risulta impossibile, in quanto nel calcolo del numero delle istruzioni viene anche considerata la riga di intestazione della classe e le righe dei commenti.

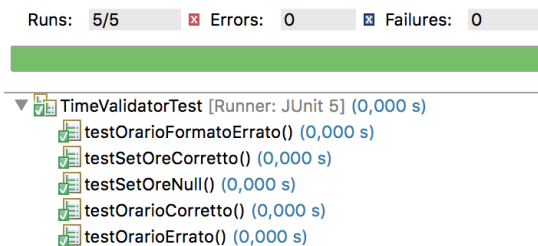


Figura 8.4: Risultato dell'esecuzione dei casi di test

| Coverage                                 |          |                      |                     |                    |
|--|----------|----------------------|---------------------|--------------------|
| TimeValidatorTest (10-ago-2018 20.51.40) |          |                      |                     |                    |
| Element                                  | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
| TimeValidator.java                       | 90,3 %   | 28                   | 3                   | 31                 |

Figura 8.5: Livello di copertura dei casi di test

### 2.3 Test della funzionalità di parsing del file XML

Nel test della funzionalità di parsing del file XML si riescono a testare diverse classi contemporaneamente:

- `DBHandler`, utilizzata per ricavare l'oggetto `EntityManager` tramite il quale eseguire delle query sul database.
- `RecordParser`, utilizzata per ottenere una `List<Timbratura>` a partire dal file XML.
- `XMLHandler`, che estende la classe `DefaultHandler` e definisce i metodi utilizzati per trattare i vari tag XML ed il loro contenuto.
- `XMLTag`, l'enumerazione contenente tutti i tag che possono essere incontrati nel file XML.

Per il test di questa funzionalità si è deciso di implementare un metodo che creasse in automatico il file XML da parsare:

```

1 void createTestXML() {
2     try {
3         DocumentBuilderFactory docFactory = DocumentBuilderFactory.
            newInstance(); DocumentBuilder docBuilder = docFactory.
            newDocumentBuilder();
4         Document doc = docBuilder.newDocument();
    }
}

```

```

5  Element records = doc.createElement("records");
6  records.setAttribute("date", "30/04/2018");
7  doc.appendChild(records);
8  Element record = doc.createElement("record");
9  record.setAttribute("type", "IN");
10 records.appendChild(record);
11 Element badgeID = doc.createElement("badgeID");
12 badgeID.appendChild(doc.createTextNode("00001"));
13 record.appendChild(badgeID);
14 Element time = doc.createElement("time");
15 time.appendChild(doc.createTextNode("00:00:00"));
16 record.appendChild(time);
17 TransformerFactory transformerFactory = TransformerFactory.
    newInstance();
18 Transformer transformer = transformerFactory.newTransformer()
    ;
19 DOMSource source = new DOMSource(doc);
20 StreamResult result = new StreamResult(new File("
    fileTestRecord.xml"));
21 transformer.transform(source, result);
22 } catch (ParserConfigurationException pce) {
23 pce.printStackTrace();
24 } catch (TransformerException tfe) {
25 tfe.printStackTrace();
26 }
27 }

```

Codice 8.3: Metodo per la creazione del file XML di test

Nei casi di test, eseguiti tramite JUnit sono state testate due situazioni:

- Parsing di un file non esistente
- Parsing di un file XML corretto

Il codice della classe di test è riportato in Codice 8.4.

```

1  package gestioneOrariTest;
2
3  import static org.junit.jupiter.api.Assertions.*;
4
5  import java.io.File;
6
7  import javax.xml.parsers.DocumentBuilder;
8  import javax.xml.parsers.DocumentBuilderFactory;
9  import javax.xml.parsers.ParserConfigurationException;
10 import javax.xml.transform.Transformer;
11 import javax.xml.transform.TransformerException;
12 import javax.xml.transform.TransformerFactory;
13 import javax.xml.transform.dom.DOMSource;
14 import javax.xml.transform.stream.StreamResult;
15 import org.junit.jupiter.api.Test;
16 import org.w3c.dom.Document;
17 import org.w3c.dom.Element;
18
19 import gestioneOrariApplication.RecordParser;
20
21 class XMLTest {
22
23     @Test
24     void testPercorsoNonEsistente() {
25         RecordParser rp = new RecordParser();

```

```

26  assertNull(rp.parserXML("1234"));
27  }
28
29  @Test
30  void fileCorretto() {
31      RecordParser rp = new RecordParser();
32      createTestXML();
33      assertNotNull(rp.parserXML("fileTestRecord.xml"));
34  }
35
36  }

```

Codice 8.4: Casi JUnit per il test delle funzionalità di gestione del file XML

Tutti i casi di test hanno dato esito positivo (Figura 8.6) ed hanno permesso di raggiungere un coverage delle classi molto buona (figura 8.7).



Figura 8.6: Risultato dell'esecuzione dei casi di test

| Coverage                                 |          |                      |                     |                    |
|--|----------|----------------------|---------------------|--------------------|
| TimeValidatorTest (11-ago-2018 11.51.49) |          |                      |                     |                    |
| Element                                  | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
| XMLHandler.java                          | 95,9 %   | 189                  | 8                   | 197                |
| XMLTag.java                              | 93,8 %   | 75                   | 5                   | 80                 |
| DBHandler.java                           | 70,0 %   | 7                    | 3                   | 10                 |
| RecordParser.java                        | 100,0 %  | 35                   | 0                   | 35                 |

Figura 8.7: Livello di copertura dei casi di test

## 2.4 Test della classe RecordAggregator

Per testare la classe `RecordAggregator`, utilizzata nel processo di aggregazione degli orari, passando dalle singole timbrature "grezze" a quelle elaborate, si è scelto di analizzare i seguenti casi:

- Nessuna timbratura grezza inserita in database
- Timbrature grezze presenti in database
- Singoli metodi chiamati passando delle date nulle

I casi di test sono stati eseguiti tramite `JUnit` ed il codice riportato in Codice 8.5.

```

1  package gestioneOrariTest;
2

```



```
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import java.sql.Time;
6 import java.util.Date;
7
8 import javax.persistence.TypedQuery;
9
10 import org.junit.jupiter.api.Test;
11
12 import gestioneOrariApplication.DBHandler;
13 import gestioneOrariApplication.RecordAggregator;
14 import gestioneOrariData.Dipendente;
15 import gestioneOrariData.Timbratura;
16
17 class RecordAggregatorTest {
18
19     @Test
20     void testTimbratureNonPresenti() {
21         RecordAggregator ra = new RecordAggregator();
22         ra.eliminaDatiTimbrature();
23         ra.aggregate();
24     }
25
26     @Test
27     void testMetodiConParametriNull() {
28         RecordAggregator ra = new RecordAggregator();
29         assertThrows(NullPointerException.class, () -> {
30             ra.elaboraOrari(null, null);
31         });
32         assertThrows(NullPointerException.class, () -> {
33             ra.creaRigheVuoteOrari(null, null);
34         });
35         assertThrows(Exception.class, () -> {
36             ra.posizionaOrari(null, null);
37         });
38     }
39
40     @SuppressWarnings("deprecation")
41     @Test
42     void testTimbraturaCorretta() {
43         Timbratura t = new Timbratura();
44         t.setData(new Date(2018-1900,12-1,20+1));
45         t.setOra(new Time(10+1,20,00));
46         t.setTipo("IN");
47         TypedQuery<Dipendente> query = DBHandler.getEntityManager().
48             createQuery("SELECT d FROM Dipendente d WHERE d.IDBadge
49                 ='00001'",Dipendente.class);
50         if (query.getResultList().isEmpty() || query.getResultList().
51             size()>1) assert(false);
52         t.setDipendente(query.getResultList().get(0));
53
54         DBHandler.getEntityManager().getTransaction().begin();
55         DBHandler.getEntityManager().persist(t);
56         DBHandler.getEntityManager().getTransaction().commit();
57         RecordAggregator ra = new RecordAggregator();
58         ra.aggregate();
59     }
60
61     @SuppressWarnings("deprecation")
62     @Test
63     void testTimbratureCorrette() {
64         Timbratura t = new Timbratura();
```

```

62 Timbratura t1 = new Timbratura();
63 Timbratura t2 = new Timbratura();
64 Timbratura t3 = new Timbratura();
65
66 t.setData(new Date(2018-1900,12-1,20+1));
67 t.setOra(new Time(10+1,20,00));
68 t.setTipo("IN");
69
70 t1.setData(new Date(2018-1900,12-1,20+1));
71 t1.setOra(new Time(13+1,20,00));
72 t1.setTipo("OUT");
73
74 t2.setData(new Date(2018-1900,12-1,20+1));
75 t2.setOra(new Time(15+1,20,00));
76 t2.setTipo("IN");
77
78 t3.setData(new Date(2018-1900,12-1,20+1));
79 t3.setOra(new Time(20+1,20,00));
80 t3.setTipo("OUT");
81
82 TypedQuery<Dipendente> query = DBHandler.getEntityManager().
    createQuery("SELECT d FROM Dipendente d WHERE d.IDBadge
    = '00001'", Dipendente.class);
83 if (query.getResultList().isEmpty() || query.getResultList().
    size() > 1) assert(false);
84
85 t.setDipendente(query.getResultList().get(0));
86 t1.setDipendente(query.getResultList().get(0));
87 t2.setDipendente(query.getResultList().get(0));
88 t3.setDipendente(query.getResultList().get(0));
89
90 DBHandler.getEntityManager().getTransaction().begin();
91 DBHandler.getEntityManager().persist(t);
92 DBHandler.getEntityManager().persist(t1);
93 DBHandler.getEntityManager().persist(t2);
94 DBHandler.getEntityManager().persist(t3);
95 DBHandler.getEntityManager().getTransaction().commit();
96 RecordAggregator ra = new RecordAggregator();
97 ra.aggregate();
98 }
99
100 }

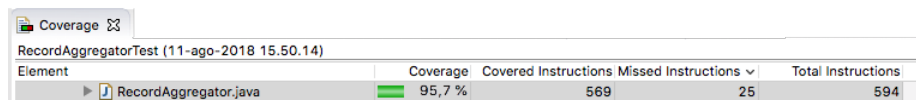
```

Codice 8.5: Casi JUnit per il test della classe RecordAggregator

Tutti i casi di test hanno dato esito positivo (Figura 8.8) ed hanno permesso di raggiungere un coverage della classe **RecordAggregator** dell'95.7% delle istruzioni (figura 8.9).



Figura 8.8: Risultato dell'esecuzione dei casi di test



The screenshot shows a code coverage tool window titled 'Coverage' with a search icon. Below the title bar, it says 'RecordAggregatorTest (11-ago-2018 15.50.14)'. A table displays the coverage data for 'RecordAggregator.java'.

| Element               | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
|-----------------------|----------|----------------------|---------------------|--------------------|
| RecordAggregator.java | 95,7 %   | 569                  | 25                  | 594                |

Figura 8.9: Livello di copertura dei casi di test



## Parte III

### Fase 3



# Capitolo 9

## Introduzione

### 1 Scelta della funzione da implementare

**Selezione del caso d'uso da implementare** Per questa *Fase 3* del processo di sviluppo dell'applicazione si è scelto di implementare la seconda delle due funzionalità chiave del programma, ovvero la funzionalità descritta all'interno del caso d'uso UC8: Generazione turnazione settimanale.

**Breve descrizione dell'obiettivo** Si vuole creare una procedura che, considerando tutti i dipendenti dell'azienda committente, sia in grado di allocare i turni programmati per la settimana, rispettando il numero massimo di ore che possono essere lavorate da ciascun dipendente ogni settimana e rispettando il fabbisogno per ciascuna attività.

**Dati di input** I dati di input necessari per il funzionamento della funzionalità analizzata in questa fase sono:

- Lista dei dipendenti dell'azienda committente.
  - Numero massimo di ore settimanali per ciascun dipendente.
- Lista delle attività da svolgere.
- Lista dei turni.
- Fabbisogno di dipendenti, per ciascun turno, in ciascuna data.

**Dati di output** L'output prodotto dall'algoritmo è un piano di allocazione dei turni, che rispetta i vincoli che sono stati posti.

### 2 Aggiornamento dei dati nel database

Anche per il caso d'uso UC8 sono necessari dei dati, all'interno del database, aggiuntivi rispetto a quelli inseriti nella *Fase 2*.

Nella versione finale del software, questi dati saranno inseribili da UI da parte dell'utente utilizzatore. A questo punto del processo di sviluppo, però, le

funzionalità di inserimento dati non sono ancora state implementate, quindi si è scelto di realizzare uno script SQL per l'inserimento di dati di prova (Codice 9.1).

```

1 INSERT INTO Dipendente (IDBadge, Nome, Cognome, DataNascita,
  PagaBaseOraria, OreContrattuali_Settimana, EMailAziendale)
  VALUES ("00004", "Giulio", "Verdi", "2014-04-04", 8, 40, "giulio.
  verdi@prova.it"), ("00005", "Andrea", "Bombarda", "1994-04-10",
  8, 40, "a.bombarda@studenti.unibg.it"), ("00006", "Federica", "
  Paravisi", "1994-07-19", 8, 40, "f.paravisi@studenti.unibg.it"),
  ("00007", "Majid", "Arif", "1993-11-08", 8, 40, "a.arif@studenti
  .unibg.it"), ("00008", "A", "B", "2018-01-01", 8, 40, "a.b@prova.
  it"), ("00009", "C", "D", "2018-01-01", 8, 40, "c.d@prova.it"), ("
  00010", "E", "F", "2018-01-01", 8, 40, "e.f@prova.it"), ("00011", "
  G", "H", "2018-01-01", 8, 40, "g.h@prova.it"), ("00012", "I", "L", "
  2018-01-01", 8, 40, "i.l@prova.it"), ("00013", "M", "N", "
  2018-01-01", 8, 40, "m.n@prova.it");

2
3 INSERT INTO Turno (Descrizione) VALUES ('00-08'), ('08-16'), ('
  16-24');

4
5 INSERT INTO Attivita (Nome, Descrizione) VALUES ('MNT-ELT', '
  Montaggio parti elettriche'), ('MNT-MEC', 'Montaggio parti
  meccaniche'), ('TEST', 'Test prodotti');

6
7 INSERT INTO Ruolo (IDBadge, IDAttivita) VALUES ('00001', 1), ('
  00001', 2), ('00002', 3), ('00002', 2), ('00003', 1), ('00004'
  , 1), ('00005', 1), ('00006', 2), ('00007', 3), ('00008', 2), ('00009
  ', 3), ('00010', 3), ('00011', 1), ('00012', 3), ('00013', 2), ('
  00004', 3), ('00008', 1), ('00011', 2), ('00011', 3), ('00013', 1);

8
9 INSERT INTO Fabbisogno_Dipendenti (IDAttivita, IDTurno, Data,
  N_Dipendenti) VALUES ('1', '1', '2018-01-01', '1'), ('1', '2', '
  2018-01-01', '1'), ('1', '3', '2018-01-01', '1'), ('1', '1', '
  2018-01-02', '1'), ('1', '2', '2018-04-02', '1'), ('1', '3', '
  2018-01-02', '1'), ('1', '1', '2018-01-03', '1'), ('1', '2', '
  2018-01-03', '1'), ('1', '3', '2018-01-03', '1'), ('1', '1', '
  2018-01-04', '1'), ('1', '2', '2018-01-04', '1'), ('1', '3', '
  2018-01-04', '1'), ('1', '1', '2018-01-05', '1'), ('1', '2', '
  2018-04-05', '1'), ('1', '3', '2018-01-05', '1'), ('1', '1', '
  2018-01-06', '1'), ('1', '2', '2018-01-06', '1'), ('1', '3', '
  2018-01-06', '1'), ('1', '1', '2018-01-07', '1'), ('1', '2', '
  2018-01-07', '1'), ('1', '3', '2018-01-07', '1'), ('2', '1', '
  2018-01-01', '1'), ('2', '2', '2018-01-01', '1'), ('2', '3', '
  2018-01-01', '1'), ('2', '1', '2018-01-02', '1'), ('2', '2', '
  2018-04-02', '1'), ('2', '3', '2018-01-02', '1'), ('2', '1', '
  2018-01-03', '1'), ('2', '2', '2018-01-03', '1'), ('2', '3', '
  2018-01-03', '1'), ('2', '1', '2018-01-04', '1'), ('2', '2', '
  2018-01-04', '1'), ('2', '3', '2018-01-04', '1'), ('2', '1', '
  2018-01-05', '1'), ('2', '2', '2018-04-05', '1'), ('2', '3', '
  2018-01-05', '1'), ('2', '1', '2018-01-06', '1'), ('2', '2', '
  2018-01-06', '1'), ('2', '3', '2018-01-06', '1'), ('2', '1', '
  2018-01-07', '1'), ('2', '2', '2018-01-07', '1'), ('2', '3', '
  2018-01-07', '1'), ('3', '1', '2018-01-01', '1'), ('3', '2', '
  2018-01-01', '1'), ('3', '3', '2018-01-01', '1'), ('3', '1', '
  2018-01-02', '1'), ('3', '2', '2018-04-02', '1'), ('3', '3', '
  2018-01-02', '1'), ('3', '1', '2018-01-03', '1'), ('3', '2', '
  2018-01-03', '1'), ('3', '3', '2018-01-03', '1'), ('3', '1', '
  2018-01-04', '1'), ('3', '2', '2018-01-04', '1'), ('3', '3', '
  2018-01-04', '1'), ('3', '1', '2018-01-05', '1'), ('3', '2', '
  2018-04-05', '1'), ('3', '3', '2018-01-05', '1'), ('3', '1', '
  2018-01-06', '1'), ('3', '2', '2018-01-06', '1'), ('3', '3', '
  2018-01-06', '1');

```



```
2018-01-06','1'),('3','1','2018-01-07','1'),('3','2','  
2018-01-07','1'),('3','3','2018-01-07','1');
```

Codice 9.1: Query di aggiornamento dei dati nel database



## Capitolo 10

# Progettazione algoritmo

### 1 Modello del problema

Il problema di allocazione dei dipendenti sui turni, in base al numero massimo di ore ricopribili da ciascun dipendente ed in base all'attività che può essere svolta da ciascuna persona, può essere rappresentato dal modello di programmazione lineare intera binaria descritto di seguito.

Questo problema è noto, in letteratura, con il nome di **Staff Scheduling** o **Employee Scheduling**.

#### 1.1 Indici utilizzati

Nella trattazione successiva, gli indici utilizzati a pedice, avranno i seguenti significati:

- $j$ , viene utilizzato per identificare la  $j$ -esima **attività**.
- $i$ , viene utilizzato per indicare l' $i$ -esimo **dipendente**.
- $l$ , viene utilizzato per indicare l' $l$ -esimo **giorno**.  
Nella situazione qui analizzata si ha:

$$1 \leq l \leq 7 \quad (10.1)$$

- $k$ , viene utilizzato per indicare il  $k$ -esimo **turno**.  
Nella situazione qui analizzata si ha:

$$1 \leq k \leq 3 \quad (10.2)$$

#### 1.2 Variabili decisionali

L'unica variabile decisionale utilizzata in questo problema è

$$x_{ijkl} = \begin{cases} 1 \\ 0 \end{cases} \quad (10.3)$$

con  $x_{ijkl} = 1$  se il dipendente  $i$ , per l'attività  $j$  viene assegnato al turno  $k$  nel giorno  $j$ , mentre  $x_{ijkl} = 0$  altrimenti.

### 1.3 Variabili considerate

Al fine di riuscire a stabilire il fabbisogno di dipendenti, considerando il numero massimo di ore lavorabili da ciascun dipendente, sono state individuate due ulteriori variabili importanti:

- $R_{jkl}$ , viene utilizzata per indicare il fabbisogno di dipendenti per l'attività  $j$ , nel turno  $k$  del giorno  $l$ .
- $L_i$ , viene utilizzata per indicare il numero di ore massime settimanali lavorabili da un dipendente  $i$ .
- $F_i$ , viene utilizzata per indicare il numero di ore che possono essere ancora assegnate per il dipendente  $i$ .

### 1.4 Vincoli

Per il problema dell'allocazione dei turni, si sono individuati i seguenti vincoli:

- Ogni dipendente può lavorare al massimo un turno per giorno:

$$\sum_j \sum_k x_{ijkl} \leq 1 \quad , \quad \forall i, \forall l \quad (10.4)$$

- Il fabbisogno di dipendenti in ogni attività e per ogni giorno deve essere coperto:

$$\sum_i x_{ijkl} = R_{jkl} \quad , \quad \forall i, \forall l, \forall k \quad (10.5)$$

- La somma delle ore lavorate da ciascun dipendente, nella settimana, non deve eccedere il limite di ore del singolo dipendente:

$$8 \cdot \sum_k \sum_j \sum_l x_{ijkl} \leq L_i \quad , \quad \forall i \quad (10.6)$$

### 1.5 Funzione obiettivo

La funzione obiettivo per un problema di questo tipo deve riuscire a minimizzare il numero di dipendenti che vengono fatti lavorare nella settimana considerata, in modo da non avere inutili eccedenze di personale.

In particolare, introduciamo una nuova variabile  $y_i \in \{0, 1\}$  che viene posta uguale ad 1 se l' $i$ -esimo dipendente viene fatto lavorare, a 0 altrimenti. Si ottiene in questo modo la seguente funzione obiettivo:

$$\phi = \min \sum_i y_i \quad (10.7)$$

## 2 Algoritmo greedy per la risoluzione

Per la risoluzione del problema di Staff Scheduling in oggetto, si è scelto di utilizzare l'algoritmo greedy il cui pseudocodice è riportato nel Codice 10.1. Questo tipo di algoritmo è stato mutuato (e riadattato) dal problema del MCMs (ordinamento di lavori su macchine, con minimizzazione del numero di macchine).

**Passi dell'algoritmo** L'algoritmo greedy per la risoluzione del problema di Staff Scheduling con minimizzazione del numero di operai può essere composto dai seguenti passi:

- Inizializzazione:
  - Nessun turno è assegnato
  - Nessun dipendente è selezionato per lavorare ( $y_i = 0$ )
  - Per ciascun dipendente  $F_i = L_i$
- Iterazione, per ogni slot (*turno + data + tipo di attività*) da assegnare:
  - Se lo slot è stato completamente allocato, si passa allo slot successivo.
  - Altrimenti...
  - Se lo slot può essere ricoperto da un dipendente che è già stato utilizzato nel processo di allocazione (ovvero con  $y_i = 1$  e  $F_i \geq 8$ ), si assegna lo slot a quel dipendente, e si passa all'allocazione successiva.
  - Altrimenti...
  - Tra tutti i dipendenti che sono in grado di svolgere l'attività da allocare, che non hanno lavorato già un turno lo stesso giorno dello slot considerato, e che hanno  $F_i \geq 8$  si seleziona il dipendente più promettente, in base ad una determinata euristica.

Se non è possibile trovare un'allocazione, il problema non è risolvibile, quindi non è possibile trovare nessun tipo di schedulazione.

## 2.1 Pseudocodice dell'algoritmo

```

1  algoritmo schedulaTurni (dataMin, dataMax) -> AssegnazioneTurno
2  //=====
3  // Inizializzazione
4  //=====
5  // Nessun turno e' assegnato
6  ListaAssegnazioni <- nil
7  // Tutti i dipendenti hanno le ore "rimanenti" uguale a
8  // quelle totali, nessuno dei dipendenti viene fatto
9  // lavorare
10 for each dipendente:
11   F_i <- L_i
12   y_i <- 0
13 end for
14 //=====
15 // Iterazione
16 //=====
17 // Scorrimento di tutti gli slot da assegnare, in ordine di
18 // data, di turno ed infine di attivita'
19 for each t in Fabbisogno_Dipendenti between dataMin and
   dataMax:
20   // Cerco di assegnare i dipendenti necessari
21   while R(t)>0
22     // Si e' deciso di assegnare per primi i dipendenti che
23     // possono svolgere meno attivita', in modo da avere dei
24     // "Jolly" liberi
25     ListaDipendenti <- Lista di tutti i dipendenti in grado
26     di svolgere Attivita(t) ordinata
27     per numero di attivita svolgibili in ordine crescente

```

```

28     e che hanno  $F_i \geq 8$  e che non hanno lavorato nello
29     stesso giorno
30     if inters(ListaDipendenti, ListaDipendentiAssegnati)
31         != {Insieme Vuoto} then
32         ListaAssegnazioni.add(t,
33             ListaDipendenti[indice_min_intersezione])
34          $R(t) \leftarrow R(t) - 1$ 
35          $F_{\{ListaDipendenti[indice\_min\_intersezione]\}} \leftarrow$ 
36              $F_{\{ListaDipendenti[indice\_min\_intersezione]\}} - 8$ 
37          $y_{\{ListaDipendenti[indice\_min\_intersezione]\}} \leftarrow 1$ 
38     else
39         if (ListaDipendenti.size > 0) then
40             // Aggiunta del primo dipendente
41             ListaAssegnazioni.add(t, ListaDipendenti[0])
42              $R(t) \leftarrow R(t) - 1$ 
43              $F_{\{ListaDipendenti[0]\}} \leftarrow$ 
44                  $F_{\{ListaDipendenti[0]\}} - 8$ 
45              $y_{\{ListaDipendenti[0]\}} \leftarrow 1$ 
46         else
47             SCHEDULE NON POSSIBILE!!
48             return nil
49         end if
50     end if
51 end while
52 end for
53 return ListaAssegnazioni
54 fine algoritmo

```

Codice 10.1: Pseudocodice per l'algoritmo Greedy di Staff Scheduling

**Euristica** In questo algoritmo greedy, come euristica utilizzata nell'ordinamento della lista dei dipendenti si è scelto di considerare il numero di attività che possono essere svolte da ciascun dipendente, prendendo per primi i dipendenti che possono svolgere meno attività.

In questo modo si lasciano per ultimi i vari dipendenti che possono svolgere più attività e si hanno a disposizione delle scelte "Jolly" con il diminuire delle possibilità di scelta.

## 2.2 Analisi della complessità dell'algoritmo

Analizziamo ora la complessità dell'algoritmo di Staff Scheduling, ponendosi nella situazione peggiore. Consideriamo:

- $n$ , il numero di dipendenti dell'azienda
- 7 giorni come orizzonte di pianificazione
- 3 turni da pianificare per ogni giornata
- $m$ , il numero totale di attività da pianificare ogni giorno

In questo modo possiamo fare le seguenti considerazioni:

- L'operazione di inizializzazione (righe 10-13 del Codice 10.1) viene eseguita una volta per ogni dipendente, quindi ha un costo  $O(n)$ .

- L'operazione di selezione della lista di tutti i dipendenti che sono in grado di svolgere l'attività (righe 25-29 del Codice 10.1) è scomponibile in diverse operazioni con costo unitario  $O(1)$ , un eventuale ciclo sui dipendenti con costo  $O(n)$  ed un ordinamento che, considerando un algoritmo di ordinamento ottimo, può essere considerato con costo  $O(n \cdot \log n)$ . L'intera complessità computazionale di questa sezione della funzione risulta quindi:

$$O(1) + O(n) + O(n \cdot \log n) = O(n \cdot \log n) \quad (10.8)$$

- Il ciclo for-each (riga 19 del Codice 10.1) viene eseguito, nel peggiore dei casi  $7 \cdot 3 \cdot m$  volte, quindi si ha una complessità computazionale pari a:

$$O(7 \cdot 3 \cdot m) = O(m) \quad (10.9)$$

- Il ciclo while (riga 21 del Codice 10.1) viene eseguito, nel peggiore dei casi un numero di volte pari ad un terzo del numero dei dipendenti (*caso in cui tutti i dipendenti sono in grado di svolgere tutte le attività, quindi vengono divisi equamente nella giornata*), quindi ha una complessità pari a:

$$O\left(\frac{1}{3}n\right) = O(n) \quad (10.10)$$

Unendo tutte le complessità individuate per ciascuna sezione, si arriva alla conclusione che la complessità computazionale totale dell'algoritmo greedy proposto risulta:

$$T(n) = O(n) + O(m \cdot n \cdot n \cdot \log n) = O(m \cdot n^2 \cdot \log n) \quad (10.11)$$

Con buona approssimazione, data la dimensione dell'azienda committente, è possibile considerare il numero di attività  $m$  molto inferiore al numero di dipendenti  $n$ , ovvero  $m \ll n$ . Con questa approssimazione è possibile considerare una complessità finale totale pari a:

$$T(n) \simeq O(n^2 \cdot \log n) \quad (10.12)$$

## 2.3 Flow-chart dell'algoritmo

L'algoritmo presentato nello pseudocodice 10.1 può essere visto anche sotto forma di diagramma di flusso, così come rappresentato in Figura 10.1.

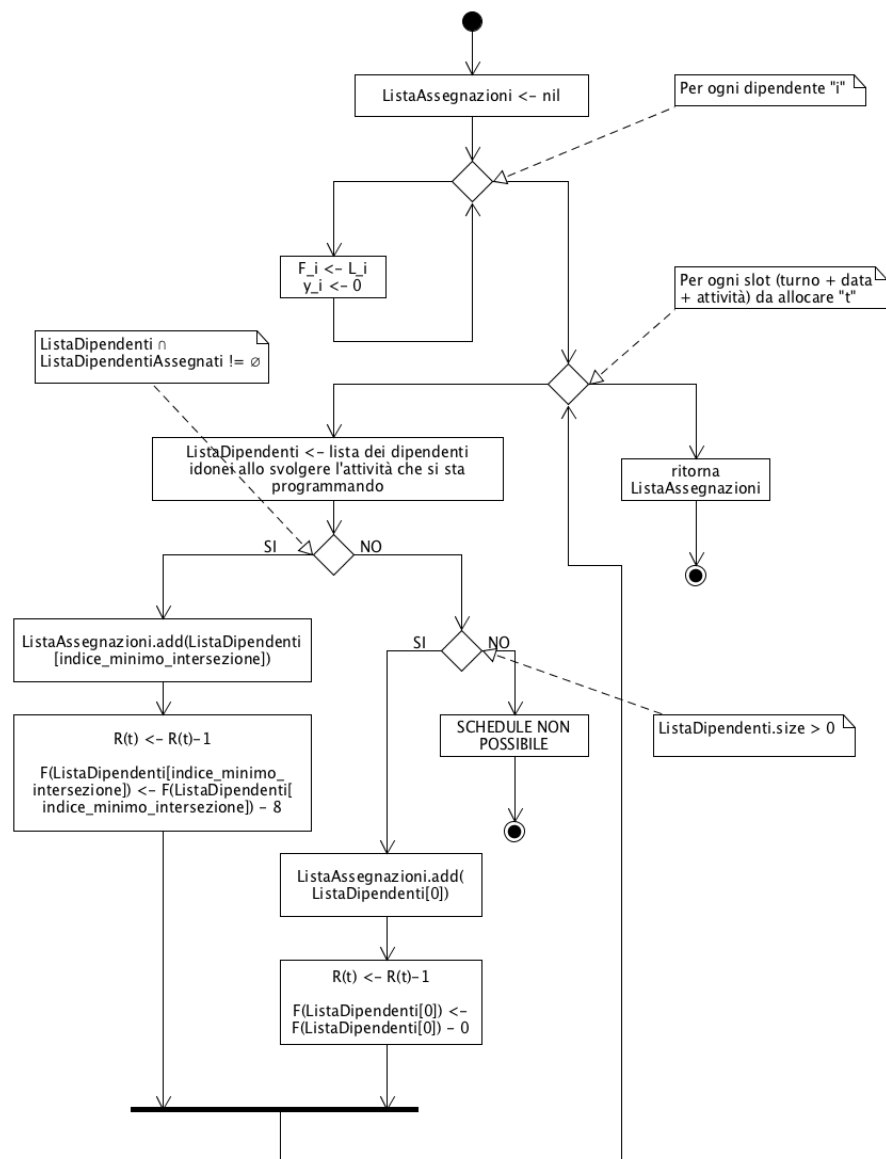


Figura 10.1: Flow chart dell'algoritmo greedy per il problema dello Staff Scheduling

## 2.4 Risultato atteso

In base ai dati inseriti manualmente nel database, ci si attende che l'algoritmo di staff scheduling non sia in grado di coprire interamente una settimana. Infatti, la schedulazione avviene correttamente fino al quinto giorno compreso, mentre il sesto giorno risulta impossibile programmare il terzo turno per la seconda



attività (MNT-MEC).

Il risultato atteso, in questo caso, è riportato in Figura 10.2.

| Badge | Giorno | Turno |
|-------|--------|-------|
| 00003 | 1      | 1     |
| 00006 | 1      | 1     |
| 00007 | 1      | 1     |
| 00005 | 1      | 2     |
| 00001 | 1      | 2     |
| 00009 | 1      | 2     |
| 00004 | 1      | 3     |
| 00002 | 1      | 3     |
| 00010 | 1      | 3     |
|       |        |       |
| 00003 | 2      | 1     |
| 00006 | 2      | 1     |
| 00007 | 2      | 1     |
| 00005 | 2      | 3     |
| 00001 | 2      | 3     |
| 00009 | 2      | 3     |
|       |        |       |
| 00003 | 3      | 1     |
| 00006 | 3      | 1     |
| 00007 | 3      | 1     |
| 00005 | 3      | 2     |
| 00001 | 3      | 2     |
| 00009 | 3      | 2     |
| 00004 | 3      | 3     |
| 00002 | 3      | 3     |
| 00010 | 3      | 3     |

| Badge  | Giorno | Turno |
|--------|--------|-------|
| 00003  | 4      | 1     |
| 00006  | 4      | 1     |
| 00007  | 4      | 1     |
| 00005  | 4      | 2     |
| 00001  | 4      | 2     |
| 00009  | 4      | 2     |
| 00004  | 4      | 3     |
| 00002  | 4      | 3     |
| 00010  | 4      | 3     |
|        |        |       |
| 00003  | 5      | 1     |
| 00006  | 5      | 1     |
| 00007  | 5      | 1     |
| 00005  | 5      | 3     |
| 00001  | 5      | 3     |
| 00009  | 5      | 3     |
|        |        |       |
| 00004  | 6      | 1     |
| 00002  | 6      | 1     |
| 00010  | 6      | 1     |
| 00008  | 6      | 2     |
| 00013  | 6      | 2     |
| 00012  | 6      | 2     |
| 00011  | 6      | 3     |
| ERRORE | 6      | 3     |

Figura 10.2: Schedule parziale per 6 giorni

## 3 Implementazione dell'algoritmo

### 3.1 Interfaccia Scheduler

Per l'implementazione dell'algoritmo di Staff Scheduling si è scelto di realizzare una interfaccia `Scheduler<T>`, che definisce il metodo `Vector<T> makeSchedule(Date dataMin, Date dataMax)`, utilizzato per generare la turnazione all'interno di un intervallo di date.

Il codice di questa interfaccia è riportato all'interno del Codice 10.2.

```

1 package gestioneTurniApplication;
2
3 import java.util.Vector;
4 import java.util.Date;
5

```

```
6 public interface Scheduler<T> {
7     public Vector<T> makeSchedule(Date dataMin, Date dataMax)
8         throws ImpossibleSchedulingException;
9 }
```

Codice 10.2: Interfaccia Scheduler

Questo metodo dichiara, tramite l'istruzione **throws** che può essere lanciata una eccezione personalizzata di tipo **ImpossibleSchedulingException**, definita come da Codice 10.3.

```
1 package gestioneTurniApplication;
2
3 public class ImpossibleSchedulingException extends Exception {
4
5     private static final long serialVersionUID = 1L;
6
7     public ImpossibleSchedulingException(String message) {
8         super(message);
9     }
10 }
```

Codice 10.3: Eccezione ImpossibleSchedulingException

### 3.2 Classe StaffScheduler

La classe **StaffScheduler** implementa l'interfaccia **Scheduler** e viene utilizzata per risolvere il problema dello Staff Scheduling.

Lo scheletro del codice di questa classe è riportato in Codice 10.4.

```
1 package gestioneTurniApplication;
2
3 public class StaffScheduler implements Scheduler<
4     AssegnazioneTurno> {
5
6     // Lista dei dipendenti utilizzati in giornata
7     private ArrayList<String> listaBadgeUtilizzati = new
8         ArrayList<String>();
9     // Data corrente
10    private Date dataCorrente = new Date();
11    // Vettore risultato, da utilizzare quando si ha un fail
12    // nello schedule
13    Vector<AssegnazioneTurno> listaResultErr = new Vector<
14        AssegnazioneTurno>();
15
16    /*
17     * Funzione per la generazione del predicato di filtro dei
18     * dipendenti
19     * ammissibili
20     */
21    private Predicate<Dipendente> getPossiblePredicate(
22        Fabbisogno_Dipendenti d) {
23        ...
24    }
25
26    /*
27     * Funzione per la generazione del predicato di filtro dei
28     * dipendenti già'
29     * assegnati
30     */
31 }
```

```
23  */
24  private Predicate<Dipendente> getAssignedPredicate() {
25      ...
26  }
27
28  /*
29  * Funzione per la generazione dello schedule
30  */
31  @Override
32  public Vector<AssegnazioneTurno> makeSchedule(Date dataMin,
33      Date dataMax) throws ImpossibleSchedulingException {
34      ...
35  }
36
37  /*
38  * Funzione che permette comunque di risalire allo schedule
39  * parziale, qualora ci sia stato un errore
40  */
41  public Vector<AssegnazioneTurno> getPartialSchedule() {
42      ..
43  }
```

Codice 10.4: Classe StaffScheduler

All'interno di questa classe sono stati inseriti:

- **Attributi:**

- **listaBadgeUtilizzati**: viene utilizzato dai metodi della classe per tenere memorizzata la lista di tutti i badge dei dipendenti che sono già stati impiegati durante la giornata.
- **dataCorrente**: viene utilizzato per la memorizzazione della data per la quale si sta generando lo schedule.
- **listaResultErr**: viene utilizzato per la memorizzazione dello schedule nel caso di errore, in modo che si riesca comunque ad accedere (nonostante l'eccezione) alle giornate schedulate correttamente.

- **Metodi:**

- **getPossiblePredicate**: metodo utilizzato per la generazione del predicato di filtro dei dipendenti ad ogni iterazione del processo di staff scheduling.  
Per ogni allocazione è necessario, infatti, considerare solamente i dipendenti che:
  1. Non hanno superato il massimo numero di ore settimanali.
  2. Non sono già stati schedulati per un turno nella giornata considerata.
  3. Sono in grado di compiere l'attività per la quale si sta generando lo schedule.
- **getAssignedPredicate**: metodo utilizzato per la generazione del predicato di filtro dei dipendenti già considerati nel corso nella settimana.  
Questo predicato è utile poichè, per minimizzare il numero di dipendenti utilizzati, è richiesto che si assegni (per quanto possibile)

la maggior parte possibile degli slot a dipendenti che sono già stati considerati.

- **makeSchedule**: metodo principale per la classe **StaffScheduler**, utilizzato per la generazione di uno schedule, per le giornate tra **dataMin** e **dataMax**.
- **getPartialSchedule**: metodo utilizzato per la restituzione dello schedule parziale, nel caso ci siano stati errori nel processo di schedulazione.

### 3.3 Classe **ListIntersect<T>**

Come riportato nel flow-chart in Figura 10.1, nell'algoritmo greedy per la risoluzione del problema di Staff Scheduling è fondamentale poter calcolare una *intersezione* tra l'insieme dei dipendenti che possono coprire un certo slot di tempo e l'insieme dei dipendenti che sono già stati assegnati.

Questo passo è proprio quello che permette di minimizzare il numero di dipendenti necessari all'azienda committente per soddisfare il fabbisogno richiesto.

A tal fine, è stata realizzata una classe **ListIntersect<T>** che implementa l'interfaccia **Intersect<T>** e che, essendo generica, permette di calcolare l'intersezione tra due qualsiasi **List** (Codice 10.5) tramite la chiamata del metodo **intersect(List<T> a, List<T> b)**.

```
1 package gestioneTurniApplication;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class ListIntersect<T> implements Intersect<T>{
7
8     /*
9      * Funzione per il calcolo dell'intersezione tra due
10     * arrayList
11     */
12     @Override
13     public ArrayList<T> intersect(List<T> a, List<T> b) {
14         ArrayList<T> lstIntersectAB = new ArrayList<T>(a);
15         lstIntersectAB.retainAll(b);
16         return lstIntersectAB;
17     }
18 }
```

Codice 10.5: Classe **ListIntersect**

## 4 Risultato ottenuto

Come riportato nella Figura 10.2, utilizzando i dati di prova inseriti all'interno del database, si deve ottenere un errore nella schedulazione.

Utilizzando la porzione di software implementata in questa fase si ottiene, infatti, il messaggio di errore riportato in Figura 10.3.

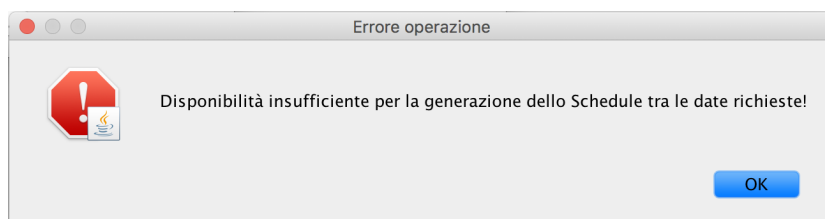


Figura 10.3: Messaggio di errore nella schedulazione

Inoltre, cliccando su **Ok**, viene mostrata comunque la schedulazione parziale che è stata realizzata, come da Figura 10.4 (coincidente con quella in Figura 10.2).

| Badge | Data       | Turno |
|-------|------------|-------|
| 00003 | 01/01/2018 | 00-08 |
| 00006 | 01/01/2018 | 00-08 |
| 00007 | 01/01/2018 | 00-08 |
| 00005 | 01/01/2018 | 08-16 |
| 00001 | 01/01/2018 | 08-16 |
| 00009 | 01/01/2018 | 08-16 |
| 00004 | 01/01/2018 | 16-24 |
| 00002 | 01/01/2018 | 16-24 |
| 00010 | 01/01/2018 | 16-24 |
| 00003 | 02/01/2018 | 00-08 |
| 00006 | 02/01/2018 | 00-08 |
| 00007 | 02/01/2018 | 00-08 |
| 00005 | 02/01/2018 | 16-24 |
| 00001 | 02/01/2018 | 16-24 |
| 00009 | 02/01/2018 | 16-24 |
| 00003 | 03/01/2018 | 00-08 |
| 00006 | 03/01/2018 | 00-08 |
| 00007 | 03/01/2018 | 00-08 |
| 00005 | 03/01/2018 | 08-16 |
| 00001 | 03/01/2018 | 08-16 |
| 00009 | 03/01/2018 | 08-16 |
| 00004 | 03/01/2018 | 16-24 |
| 00002 | 03/01/2018 | 16-24 |
| 00010 | 03/01/2018 | 16-24 |
| 00003 | 04/01/2018 | 00-08 |
| 00006 | 04/01/2018 | 00-08 |
| 00007 | 04/01/2018 | 00-08 |
| 00005 | 04/01/2018 | 08-16 |
| 00001 | 04/01/2018 | 08-16 |
| 00009 | 04/01/2018 | 08-16 |
| 00004 | 04/01/2018 | 16-24 |
| 00002 | 04/01/2018 | 16-24 |
| 00010 | 04/01/2018 | 16-24 |
| 00003 | 05/01/2018 | 00-08 |
| 00006 | 05/01/2018 | 00-08 |
| 00007 | 05/01/2018 | 00-08 |
| 00005 | 05/01/2018 | 16-24 |
| 00001 | 05/01/2018 | 16-24 |
| 00009 | 05/01/2018 | 16-24 |
| 00004 | 06/01/2018 | 00-08 |
| 00002 | 06/01/2018 | 00-08 |
| 00010 | 06/01/2018 | 00-08 |
| 00008 | 06/01/2018 | 08-16 |
| 00013 | 06/01/2018 | 08-16 |
| 00012 | 06/01/2018 | 08-16 |
| 00011 | 06/01/2018 | 16-24 |

Figura 10.4: Schedule parziale prodotto dal software



## Capitolo 11

# Test ed analisi del componente implementato

### 1 Analisi statica

Attraverso il tool **Stan4j** è stata effettuata l'analisi statica della porzione del software sviluppata in questa fase. Analizzando quanto riportato in Figura 11.1 risulta rispettata la struttura MVP che ci si era proposti di realizzare.

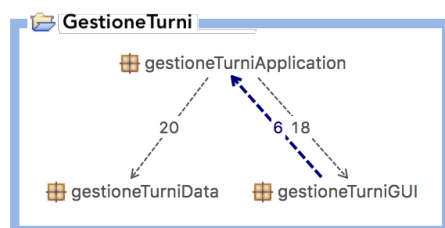


Figura 11.1: Analisi statica del codice

Dalla Figura 11.1 si nota come la componente **GUI** non comunica direttamente con la componente **Data**, ma il tutto passa attraverso la componente **Application**, proprio come previsto dal pattern MVP.

Il pattern **MVP** viene mantenuto anche includendo nell'analisi di **Stan4j** i package provenienti dalla Fase 2 (Figura 11.2).

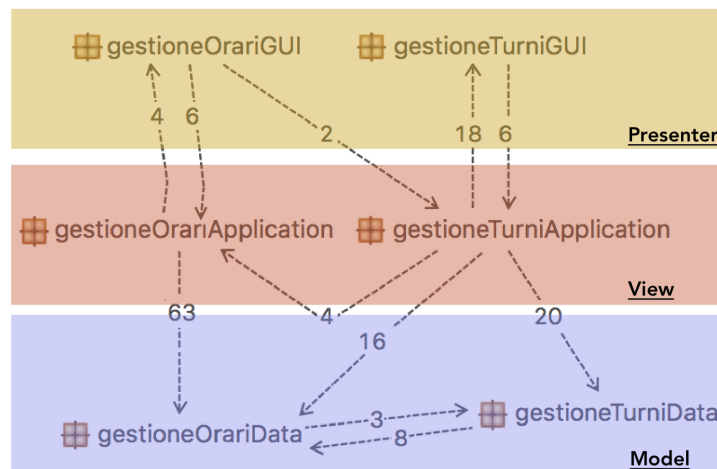


Figura 11.2: Analisi statica del codice completo

## 2 Analisi dinamica - testing

In questa sezione ci si concentrerà sull'implementazione di un certo numero di casi di test al fine di avere la massima copertura possibile del codice della porzione di software implementato in questa fase.

Data la suddivisione, tramite pattern MVP, del codice in **GUI**, **Application** e **Data** è stato scelto di testare principalmente la componente **Application**, tralasciando i quattro **Listener**, in quanto:

- La componente **GUI** è difficile da testare
- La componente **Data** è stata generata in automatico grazie alle API JPA.

Inoltre, al fine di mantenere la migliore organizzazione possibile, si è scelto di inserire i casi di test in un package a parte, denominato `gestioneTurniTest`.

### 2.1 Test della classe `ListIntersect`

Per testare la classe `ListIntersect`, utilizzata per calcolare la `List` risultante dall'intersezione di due `List`, si è scelto di analizzare i seguenti casi:

- Intersezione tra due `List` di `Integer` con elementi in comune.
- Intersezione tra due `List` di `Integer` senza elementi in comune.
- Intersezione tra due `List` di `String` con elementi in comune.
- Intersezione tra due `List` di `String` senza elementi in comune.
- Intersezione tra una `List` ed un `Null`.
- Intersezione tra due `Null`.



I casi di test sono stati eseguiti tramite JUnit ed il codice riportato in Codice 11.1.

```
1 package gestioneTurniTest;
2
3 import static org.junit.Assert.assertTrue;
4 import static org.junit.jupiter.api.Assertions.assertThrows;
5 import java.util.ArrayList;
6 import java.util.Arrays;
7 import org.junit.jupiter.api.Test;
8 import gestioneTurniApplication.ListIntersect;
9
10 public class ListIntersectTest {
11
12     @Test
13     void testCorrettiInteger() {
14         // =====
15         // Test dell'intersezione tra due List, passando argomenti
16         // corretti
17         // =====
18         // Test con due ArrayList di Integer con elemento in comune
19         ListIntersect<Integer> intersector = new ListIntersect<
20             Integer>();
21         ArrayList<Integer> result = intersector.intersect(Arrays.
22             asList(1, 2, 3), Arrays.asList(1, 5, 4));
23         assertTrue(result.get(0) == 1 && result.size() == 1);
24
25         // Test con due ArrayList di Integer senza elementi in
26         // comune
27         result = intersector.intersect(Arrays.asList(1, 2, 3),
28             Arrays.asList(4, 5, 6));
29         assertTrue(result.size() == 0);
30     }
31
32     @Test
33     void testCorrettiString() {
34         // =====
35         // Test dell'intersezione tra due List, passando argomenti
36         // corretti
37         // =====
38         // Test con due ArrayList di String con elemento in comune
39         ListIntersect<String> intersector = new ListIntersect<
40             String>();
41         ArrayList<String> result = intersector.intersect(Arrays.
42             asList("Ciao", "Come", "Va"),
43             Arrays.asList("Va", "Tutto", "Bene"));
44         assertTrue(result.get(0).equals("Va") && result.size() ==
45             1);
46
47         // Test con due ArrayList di String senza elementi in
48         // comune
49         result = intersector.intersect(Arrays.asList("Ciao", "Come",
50             "Va"), Arrays.asList("Tutto", "Bene", "Grazie"));
51         assertTrue(result.size() == 0);
52     }
53
54     @Test
55     void testElementiNull() {
56         // =====
57         // Test dell'intersezione tra una List ed un null
58         // =====
59     }
```

```

50
51     ListIntersect<String> intersector = new ListIntersect<
String>();
52
53     assertThrows(NullPointerException.class, () -> {
54         @SuppressWarnings("unused")
55         ArrayList<String> result = intersector.intersect(Arrays.
asList("Ciao", "Come", "Va"), null);
56     });
57
58     assertThrows(NullPointerException.class, () -> {
59         @SuppressWarnings("unused")
60         ArrayList<String> result = intersector.intersect(null,
Arrays.asList("Ciao", "Come", "Va"));
61     });
62
63     assertThrows(NullPointerException.class, () -> {
64         @SuppressWarnings("unused")
65         ArrayList<String> result = intersector.intersect(null,
null);
66     });
67 }
68
69 }

```

Codice 11.1: Casi JUnit per il test della classe ListIntersect

Tutti i casi di test hanno dato esito positivo (Figura 11.3) ed hanno permesso di raggiungere un coverage della classe `ListIntersect` del 100% delle istruzioni (figura 11.4).

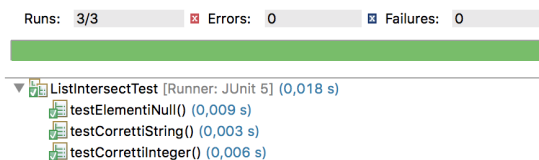


Figura 11.3: Risultato dell'esecuzione dei casi di test

| ListIntersectTest (18-ago-2018 14.35.59) |          |                       |                     |
|--|----------|-----------------------|---------------------|
| Element                                  | Coverage | Covered Instructions  | Missed Instructions |
| ListIntersect.java                       | 100,0 %  | 14                    | 0                   |
|  |          | Total Instructions 14 |                     |

Figura 11.4: Livello di copertura dei casi di test

## 2.2 Test della classe StaffScheduler

Per testare la classe `StaffScheduler`, utilizzata come implementazione dell'algoritmo greedy di Staff Scheduling. Alcuni dei metodi presenti all'interno di questa classe sono `private`, quindi risultano impossibili da testare in modo diretto. In base a queste considerazioni si è scelto di eseguire i seguenti test:

- Test del metodo `makeSchedule(Date, Date)`:
  - Test con date inserite correttamente:
    1. Con date per le quali è stato inserito un fabbisogno.
    2. Con date per le quali non è stato inserito alcun fabbisogno
    3. Con schedule possibile.
    4. Con schedule non possibile.
  - Test con date null.
  - Test con date distanti più di 7 giorni.

I casi di test sono stati eseguiti tramite JUnit ed il codice riportato in Codice 11.2.

```
1 package gestioneTurniTest;
2
3 import static org.junit.Assert.assertTrue;
4 import static org.junit.jupiter.api.Assertions.assertThrows;
5 import java.util.Date;
6 import org.junit.jupiter.api.Test;
7
8 import gestioneTurniApplication.ImpossibleSchedulingException;
9 import gestioneTurniApplication.StaffScheduler;
10
11 public class StaffSchedulerTest {
12
13     @SuppressWarnings("deprecation")
14     @Test
15     void testMakeSchedule() {
16         StaffScheduler s = new StaffScheduler();
17
18         // Test con date corrette, ma fabbisogno non inserito
19         try {
20             assertTrue(s.makeSchedule(new Date(2018 - 1900, 2, 10),
21 new Date(2018 - 1900, 2, 12)).size() == 0);
22         } catch (ImpossibleSchedulingException e) {
23             e.printStackTrace();
24         }
25
26         // Test con date corrette, con fabbisogno inserito e
27         // schedule possibile
28         try {
29             assertTrue(s.makeSchedule(new Date(2018 - 1900, 0, 1),
30 new Date(2018 - 1900, 0, 3)).size() > 0);
31         } catch (ImpossibleSchedulingException e) {
32             e.printStackTrace();
33         }
34
35         // Test con date corrette, con fabbisogno inserito ma
36         // schedule non possibile
37         try {
38             assertTrue(s.makeSchedule(new Date(2018 - 1900, 0, 1),
39 new Date(2018 - 1900, 0, 7)) == null);
40         } catch (ImpossibleSchedulingException e) {
41             e.printStackTrace();
42         }
43
44         // Test con date distanti tra di loro piu' di 7 giorni
45         try {
46             assertTrue(s.makeSchedule(new Date(2018 - 1900, 0, 1),
47 new Date(2018 - 1900, 0, 10)) == null);
48         }
49     }
50 }
```

```

42     } catch (ImpossibleSchedulingException e) {
43         e.printStackTrace();
44     }
45
46     // Test con date null
47     try {
48         s.makeSchedule(null, null);
49     } catch (NullPointerException e) {
50         e.printStackTrace();
51     } catch (ImpossibleSchedulingException e) {
52         e.printStackTrace();
53     }
54 }
55
56 @SuppressWarnings("deprecation")
57 @Test
58 void testGetPartialSchedule() {
59     StaffScheduler s = new StaffScheduler();
60
61     // Test con date corrette, con fabbisogno inserito ma
62     // schedule non possibile ->
63     // Si puo' ricavare lo schedule parziale
64     try {
65         assertTrue(s.makeSchedule(new Date(2018 - 1900, 0, 1),
66             new Date(2018 - 1900, 0, 7)) == null);
67     } catch (ImpossibleSchedulingException e) {
68         e.printStackTrace();
69     }
70     assertTrue(s.getPartialSchedule().size() > 0);
71 }

```

Codice 11.2: Casi JUnit per il test della classe StaffScheduler

Tutti i casi di test hanno dato esito positivo (Figura 11.5) ed hanno permesso di raggiungere un coverage della classe `ListIntersect` del 99.4% delle istruzioni (figura 11.6).

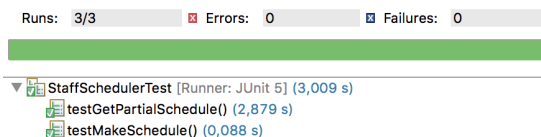


Figura 11.5: Risultato dell'esecuzione dei casi di test

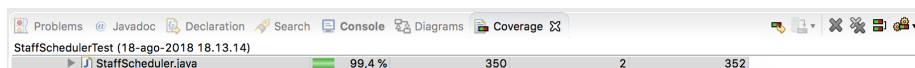


Figura 11.6: Livello di copertura dei casi di test

# Elenco delle figure

|      |  |     |
|------|--|-----|
| 2.1  | Use case diagram . . . . .   | 43  |
| 3.1  | Architettura hardware . . . . .  | 45  |
| 3.2  | Deployment diagram . . . . .   | 46  |
| 3.3  | Component diagram . . . . .  | 47  |
| 3.4  | Pattern Model-View-Presenter . . . . .   | 48  |
| 3.5  | Modello del database . . . . .   | 49  |
| 3.6  | Component diagram del componente di gestione degli orari . . . . .                 | 51  |
| 3.7  | Class diagram della GUI del componente di gestione degli orari . . . . .           | 52  |
| 3.8  | Component diagram del componente Application . . . . .                             | 53  |
| 3.9  | Class diagram del componente data del componente di gestione degli orari . . . . . | 54  |
| 3.10 | Component diagram del componente di gestione dei turni . . . . .                   | 55  |
| 3.11 | Class diagram della GUI del componente di gestione dei turni . . . . .             | 56  |
| 3.12 | Component diagram del componente Application . . . . .                             | 57  |
| 3.13 | Class diagram del componente data del componente di gestione dei turni . . . . .   | 58  |
| 6.1  | Finestra per la modifica degli orari . . . . .                                     | 72  |
| 7.1  | Flow-chart per la funzione di creazione delle righe vuote . . . . .                | 74  |
| 7.2  | Flow-chart per la funzione di posizionamento degli orari . . . . .                 | 76  |
| 7.3  | Flow-chart per la funzione di elaborazione degli orari . . . . .                   | 78  |
| 8.1  | Analisi statica del codice . . . . .   | 81  |
| 8.2  | Risultato dell'esecuzione dei casi di test . . . . .                               | 84  |
| 8.3  | Livello di copertura dei casi di test . . . . .                                    | 84  |
| 8.4  | Risultato dell'esecuzione dei casi di test . . . . .                               | 86  |
| 8.5  | Livello di copertura dei casi di test . . . . .                                    | 86  |
| 8.6  | Risultato dell'esecuzione dei casi di test . . . . .                               | 88  |
| 8.7  | Livello di copertura dei casi di test . . . . .                                    | 88  |
| 8.8  | Risultato dell'esecuzione dei casi di test . . . . .                               | 90  |
| 8.9  | Livello di copertura dei casi di test . . . . .                                    | 91  |
| 10.1 | Flow chart dell'algoritmo greedy per il problema dello Staff Scheduling . . . . .  | 104 |
| 10.2 | Schedule parziale per 6 giorni . . . . .   | 105 |
| 10.3 | Messaggio di errore nella schedulazione . . . . .                                  | 109 |
| 10.4 | Schedule parziale prodotto dal software . . . . .                                  | 109 |

|      |  |     |
|------|--|-----|
| 11.1 | Analisi statica del codice . . . . .                 | 111 |
| 11.2 | Analisi statica del codice completo . . . . .        | 112 |
| 11.3 | Risultato dell'esecuzione dei casi di test . . . . . | 114 |
| 11.4 | Livello di copertura dei casi di test . . . . .      | 114 |
| 11.5 | Risultato dell'esecuzione dei casi di test . . . . . | 116 |
| 11.6 | Livello di copertura dei casi di test . . . . .      | 116 |

# Elenco delle tabelle

|     |   |    |
|-----|---|----|
| 2.1 | Funzionalità per la gestione dei dipendenti . . . . .             | 14 |
| 2.2 | Funzionalità per la gestione dei dati base dell'azienda . . . . . | 14 |
| 2.3 | Funzionalità per la gestione dei turni dei dipendenti . . . . .   | 15 |
| 2.4 | Funzionalità per la gestione dei file XML . . . . .               | 15 |
| 2.5 | Funzionalità per la gestione degli orari dei dipendenti . . . . . | 16 |