Documentazione progetto Ingegneria del Software

Tomasoni Nicola

Sommario

Sommario **2**
Requisiti **3**
Analisi dei requisiti di sistema 3
Casi d'uso 3
Visualizza lavoratore 4
Modificare le anagrafiche 4
Aggiungere un nuovo lavoro 4
Eliminare un lavoro esistente 5
Aggiungere un nuovo lavoratore 5
Effettuare una ricerca sui lavoratori 6

Diagramma delle attività **7**
Note sullo sviluppo del sistema **8**
Processo di sviluppo 8
Design Pattern utilizzati 8
Pattern MVC 8
Pattern DAO 9
Package Model 11
Classe DaoPattern 12
Classe databaseDAOimpl 12
Classe ricerca 13
Classe persona 14
Package Controller 14
Classe Controller 15
Classe interfacciaLavoratore 15
Classe nuoviDati 15
Controlli implementati 16
Campi obbligatori 16
Email e numero di telefono validi 16
Date generiche 16
Date di nascita 16
Periodi di disponibilità 16
Set di valori predefiniti 16
Valori duplicati 16
Retribuzione valida 16
Periodo di lavoro 16

Attività di test **16**
Test dello sviluppatore 17
Test tramite peer-review 17

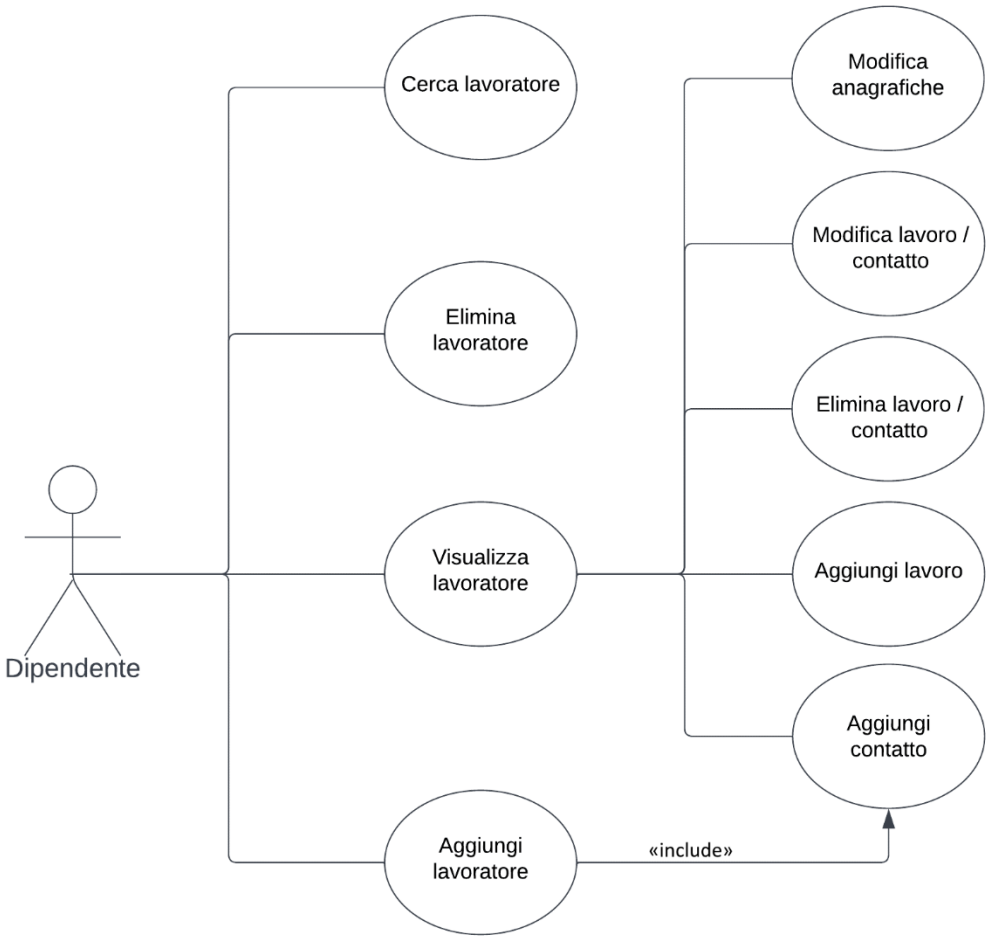
Requisiti

Analisi dei requisiti di sistema

L'applicazione sviluppata deve offrire supporto per la gestione e la ricerca di lavoratori stagionali, permettendo agli utenti del sistema di effettuare ricerche complesse per i parametri specificati dall'utente sui lavoratori iscritti al servizio. Per accedere al servizio è necessaria l'autenticazione da parte dei dipendenti dell'agenzia che offre il servizio (tramite credenziali pre-fornite dall'amministratore di sistema). Una volta all'interno del sistema sarà possibile effettuare ricerche per i lavoratori registrati, modificare i dati dei lavoratori e registrare nuovi lavoratori.

Casi d'uso

Dopo opportuna autenticazione, il dipendente ha accesso all'interfaccia che permette di interagire con il sistema.



Visualizza lavoratore
Dall'elenco dei lavoratori visualizzato dopo l'autenticazione è possibile accedere alla vista dettagliata di ogni lavoratore presente nel sistema. Dalla vista dettagliata sono disponibili tutte le informazioni anagrafiche del lavoratore più quelle relative alle sue esperienze precedenti e ai suoi contatti d'emergenza.

Precondizioni: il dipendente deve essersi autenticato
Passi:
1. Il dipendente accede al sistema
2. Il dipendente accede alla vista dettagliata per il lavoratore desiderato

Postcondizioni: viene visualizzata la vista dettagliata per il lavoratore selezionato

Modificare le anagrafiche
Dopo aver selezionato un lavoratore entrando nella sua vista dettagliata è possibile modificare tutte le informazioni precedentemente inserite.

Precondizioni:
1. Il dipendente deve essersi autenticato
2. Il dipendente deve aver selezionato un lavoratore

Passi:
3. Il dipendente accede al sistema
4. Il dipendente seleziona un lavoratore dall'elenco
5. Il dipendente accede alla schermata di modifica per le anagrafiche
6. Il dipendente modifica le voci con i nuovi valori
7. Il dipendente conferma le modifiche

Postcondizioni: le modifiche ai dati del lavoratore sono inserite

Al punto (5) verranno visualizzati tutti i campi disponibili per la modifica inizializzati con il valore attuale.
Al punto (7), in seguito alla conferma dei cambiamenti, il sistema farà un controllo dei dati inseriti dall'utente verificando che siano accettabili. Qualunque errore verrà notificato all'utente impendendo di applicare le modifiche.
In alternativa al punto (7) è possibile uscire dalla schermata di modifica senza applicare nessun cambiamento annullando la modifica.

Tale processo per la modifica delle anagrafiche è identico a quello per la modifica dei lavori e dei contatti d'emergenza, per cui non verranno trattati.

Aggiungere un nuovo lavoro
Dopo aver selezionato un lavoratore entrando nella sua vista dettagliata è possibile aggiungere un nuovo lavoro per il lavoratore selezionato.

Precondizioni:
1. Il dipendente deve essersi autenticato
2. Il dipendente deve aver selezionato un lavoratore

Passi:
3. Il dipendente accede al sistema
4. Il dipendente seleziona un lavoratore dall'elenco

5. Il dipendente accede alla schermata per la creazione di un nuovo lavoro
6. Il dipendente inserisce i nuovi dati
7. Il dipendente conferma i dati inseriti

Postcondizioni: viene aggiunto al lavoratore un nuovo lavoro.

Al punto (7) il sistema farà un controllo dei dati inseriti dall'utente verificando che siano accettabili. Qualunque errore verrà notificato all'utente impendendo la creazione di un nuovo oggetto.
In alternativa al punto (7) è possibile uscire dalla schermata di creazione senza applicare nessun cambiamento annullando la creazione.

Tale processo per la creazione di un nuovo lavoro è identico a quello per la creazione di un nuovo contatto d'emergenza, per cui non verrà trattato.

Eliminare un lavoro esistente

Dopo aver selezionato un lavoratore entrando nella sua vista dettagliata è possibile eliminare un lavoro pre esistente per il lavoratore selezionato.

Precondizioni:

1. Il dipendente deve essersi autenticato
2. Il dipendente deve aver selezionato un lavoratore

Passi:

3. Il dipendente accede al sistema
4. Il dipendente seleziona un lavoratore dall'elenco
5. Il dipendente seleziona un lavoro da eliminare
6. Il dipendente conferma l'eliminazione

Postcondizioni: il lavoro selezionato viene eliminato.

Tale processo per l'eliminazione di un lavoro è identico a quello per l'eliminazione di un contatto d'emergenza e per l'eliminazione di un lavoratore, per cui non verranno trattati. Unica differenza, il sistema non eliminerà un contatto d'emergenza se questo è l'unico che il lavoratore possiede.

Aggiungere un nuovo lavoratore

Dall'elenco dei lavoratori è possibile inserire i dati per un nuovo lavoratore da inserire nel sistema. Una volta confermati i nuovi dati anagrafici per il lavoratore vengono richiesti anche i dati per un nuovo contatto d'emergenza (dato che almeno un contatto è obbligatorio). L'utente viene quindi reindirizzato alla vista dettagliata del nuovo lavoratore.

Precondizioni: il dipendente deve essersi autenticato

Passi:

1. Il dipendente accede al sistema
2. Il dipendente accede alla schermata per la creazione di un nuovo lavoratore
3. Il dipendente inserisce i dati per un nuovo lavoratore
4. Il dipendente conferma i dati inseriti
5. Il dipendente inserisce i dati per un nuovo contatto d'emergenza
6. Il dipendente conferma i dati inseriti

Postcondizioni:

1. Il nuovo lavoratore viene aggiunto al sistema
2. Viene visualizzata la vista dettagliata per il nuovo lavoratore

Al punto (4) e al punto (6) il sistema farà un controllo dei dati inseriti dall'utente verificando che siano accettabili. Qualunque errore verrà notificato all'utente impendendo di proseguire al passo successivo.
In alternativa ai punti (4) e (6) è possibile uscire dalla schermata di creazione senza applicare nessun cambiamento annullando la creazione. In entrambi i casi l'utente verrà riportato alla schermata di elenco dei lavoratori.

Una volta terminata la creazione di un nuovo lavoratore sarà possibile dalla sua vista dettagliata modificare i dati inseriti per i campi esistenti e creare uno o più nuovi lavori o contatti d'emergenza per il nuovo lavoratore.

Effettuare una ricerca sui lavoratori
Dall'elenco dei lavoratori è possibile fare una ricerca per uno o più campi tra quelli disponibili (nome, cognome, lingue parlate, periodo di disponibilità, mansioni svolte, luogo di residenza, disponibilità di patente e auto). Per i valori specificati dall'utente il sistema offre la possibilità di effettuare ricerche AND (tutte le condizioni specificate devono essere vere) oppure ricerche OR (almeno una condizione tra quelle specificate deve essere vera). Una volta confermati i parametri della ricerca verranno visualizzati nell'elenco solo i lavoratori che corrispondono ai parametri specificati.

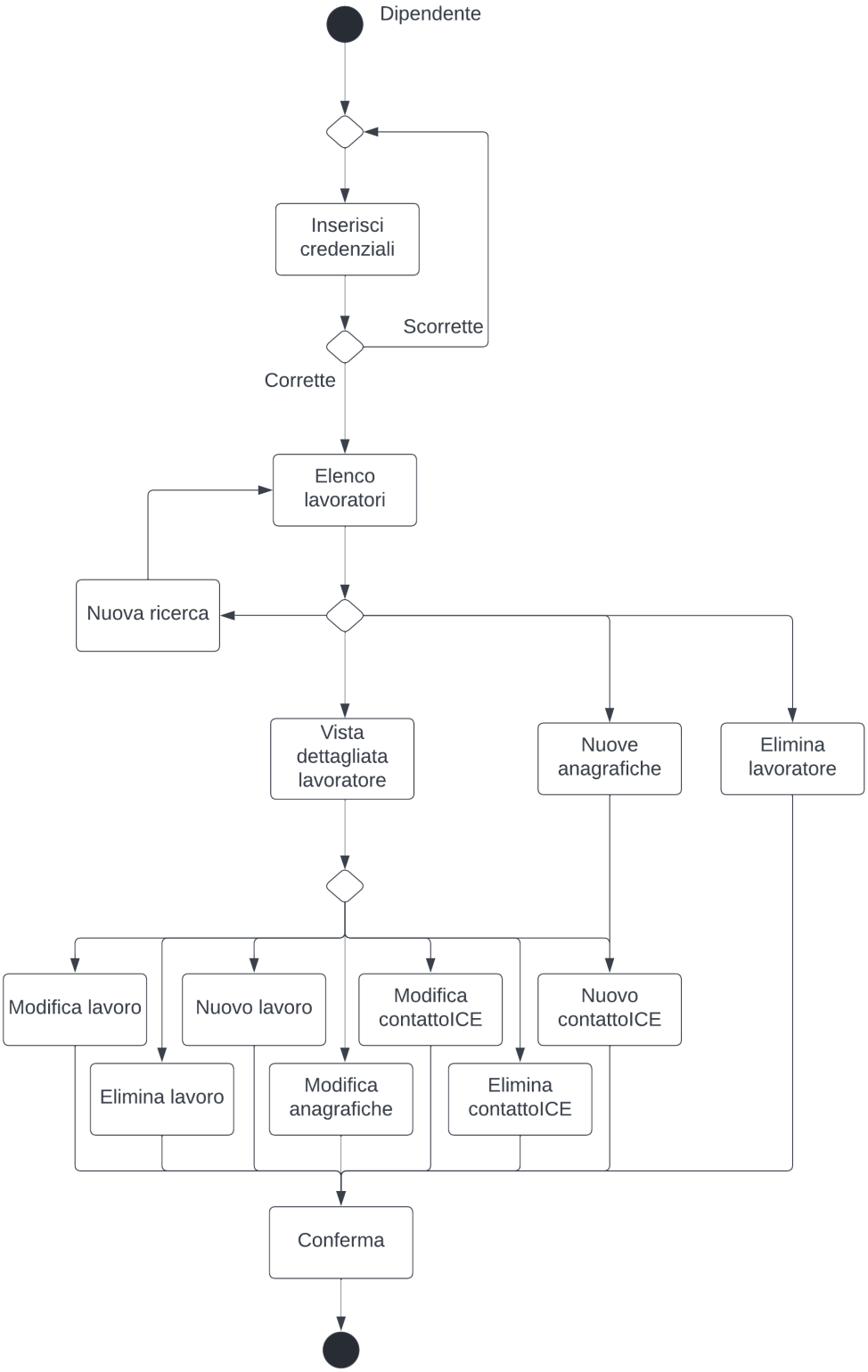
Precondizioni: il dipendente deve essersi autenticato
Passi:
1. Il dipendente accede al sistema
2. Il dipendente inserisce i parametri per una nuova ricerca
3. Il dipendente conferma i parametri selezionando il tipo di ricerca (AND/OR)

Postcondizioni: vengono visualizzati i lavoratori che rispettano i parametri

Al punto (3) il sistema farà un controllo dei parametri inseriti dall'utente verificando che siano accettabili. Qualunque errore verrà notificato all'utente impendendo di proseguire al passo successivo.

Al termine della ricerca è possibile accedere alla vista dettagliata dei lavoratori che rispettano i parametri specificati e tornare all'elenco senza annullare gli effetti della ricerca. Per visualizzare di nuovo l'elenco di tutti i lavoratori è sufficiente effettuare una nuova ricerca eliminando i parametri specificati precedentemente.

Diagramma delle attività



Note sullo sviluppo del sistema

Processo di sviluppo

Si è deciso di realizzare il sistema richiesto tramite la libreria JavaFX. Il processo di sviluppo utilizzato è stato di tipo agile e incrementale, e per ogni funzione da aggiungere al sistema si è considerato un ciclo di quattro fasi applicato in modo iterativo per ogni nuova funzione da implementare:

1. Progettazione: si è prodotta una piccola quantità di documentazione per ogni nuova funzionalità sostanziale aggiunta al sistema, principalmente diagrammi di flusso ed elenchi delle criticità. Una fase di progettazione più approfondita era già stata svolta prima dell'inizio dello sviluppo incrementale per definire i requisiti, i casi d'uso e i diagrammi UML, mentre all'interno del ciclo sono state progettate le singole funzioni più nel dettaglio.
2. Implementazione: vengono definite le funzioni da aggiungere al sistema in accordo con quanto specificato in fase di progettazione.
3. Test e validazione: alla fine dell'implementazione di ogni nuova funzione si procede con una fase di test della funzione stessa e del software per intero al fine di verificarne la robustezza. Sono verificate sia le funzionalità di base in casi standard sia in casi limite.
4. Refactoring: verificato il funzionamento di una funzionalità è prevista un'ultima fase di refactoring al fine di rendere il codice più leggibile in vista di eventuali modifiche future. Le porzioni più complesse vengono commentate.

Lo sviluppo incrementale ha permesso di poter testare il codice funzione per funzione e avendo disponibile un programma eseguibile alla fine di ogni ciclo. I cicli si sono ripetuti fino al raggiungimento dei requisiti richiesti.

Design Pattern utilizzati

Il sistema nel suo insieme è stato sviluppato secondo il pattern MVC (Model-View-Controller), implementato dividendo in tre diversi package le classi dell'applicazione. La scelta è giustificata dalla possibilità di aver potuto separare, nella fase iniziale, lo sviluppo del Model da quello View-Controller. Per motivi pratici ho infatti scelto l'interfaccia grafica come ultimo aspetto da implementare, preferendo stabilire una solida gestione dei dati prima di implementare la visualizzazione grafica degli stessi. Alla fine di questa prima parte dello sviluppo il package Model era robusto e definito e si è potuto procedere allo sviluppo parallelo del package View e del package Controller.

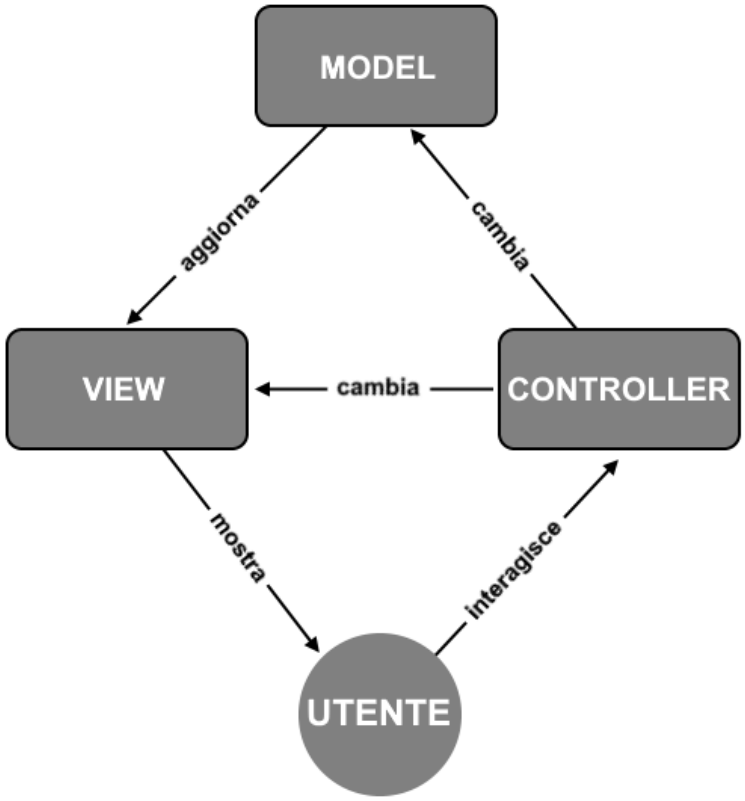
Oltre al pattern MVC è stato tuttavia implementato anche il pattern DAO (Data Access Object), più precisamente è stato implementato all'interno del package Model definito dal pattern MVC. Questo design pattern di tipo strutturale ha permesso di tenere le operazioni di import dei dati da database separate dalle funzionalità di più alto livello, nascondendone la complessità dietro all'interfaccia databaseDAO. Più nel dettaglio:

Pattern MVC

1. **Model**: all'interno del package Model sono inserite tutte le classi che permettono la gestione dei dati memorizzati. Alla fine del suo sviluppo il programma era già completo di tutte le sue funzioni, seppur accessibili solo tramite console. Durante l'implementazione del lato grafico sono nate nuove esigenze che richiedevano modifiche non sostanziali delle classi del Model, e sono state implementate in seguito senza effettuare modifiche strutturali.

2. **View**: Il package View è quello che definisce la rappresentazione grafica del Modello ed è stato implementato tramite la libreria JavaFX. Al suo interno sono quindi presenti i file FXML che definiscono il lato "statico" dell'implementazione grafica, mentre tutte le rappresentazioni che dipendono dai dati inseriti e dall'interazione con l'utente sono stati definiti dinamicamente dai relativi controller del package Controller.

3. **Controller**: Quest'ultima parte del sistema si occupa di definire il comportamento del sistema in base agli input ricevuti dall'utente. Nelle classi del package Controller sono quindi definiti i listener per le azioni dell'utente e l'implementazione della grafica dinamica (come l'elenco dei lavoratori che dipende da quali e quanti lavoratori sono memorizzati dal sistema).



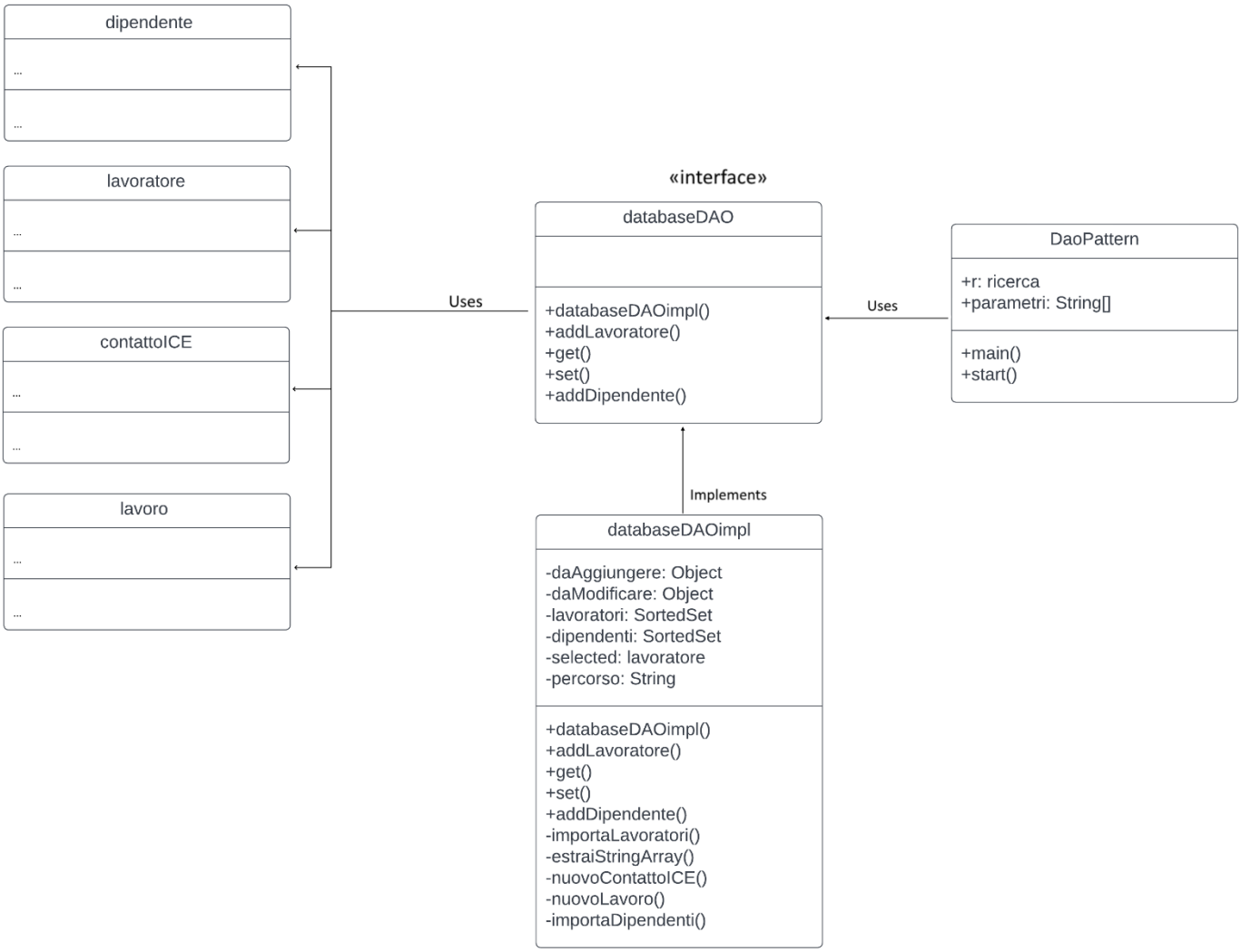
(Figura) Diagramma pattern MVC

Pattern DAO

1. **databaseDAO**: all'interno di questa interfaccia vengono definiti i metodi per la gestione dei dati di dipendenti e lavoratori, quali metodi *get*() e *set*() e le funzioni *addDipendente*() e addLavoratore().

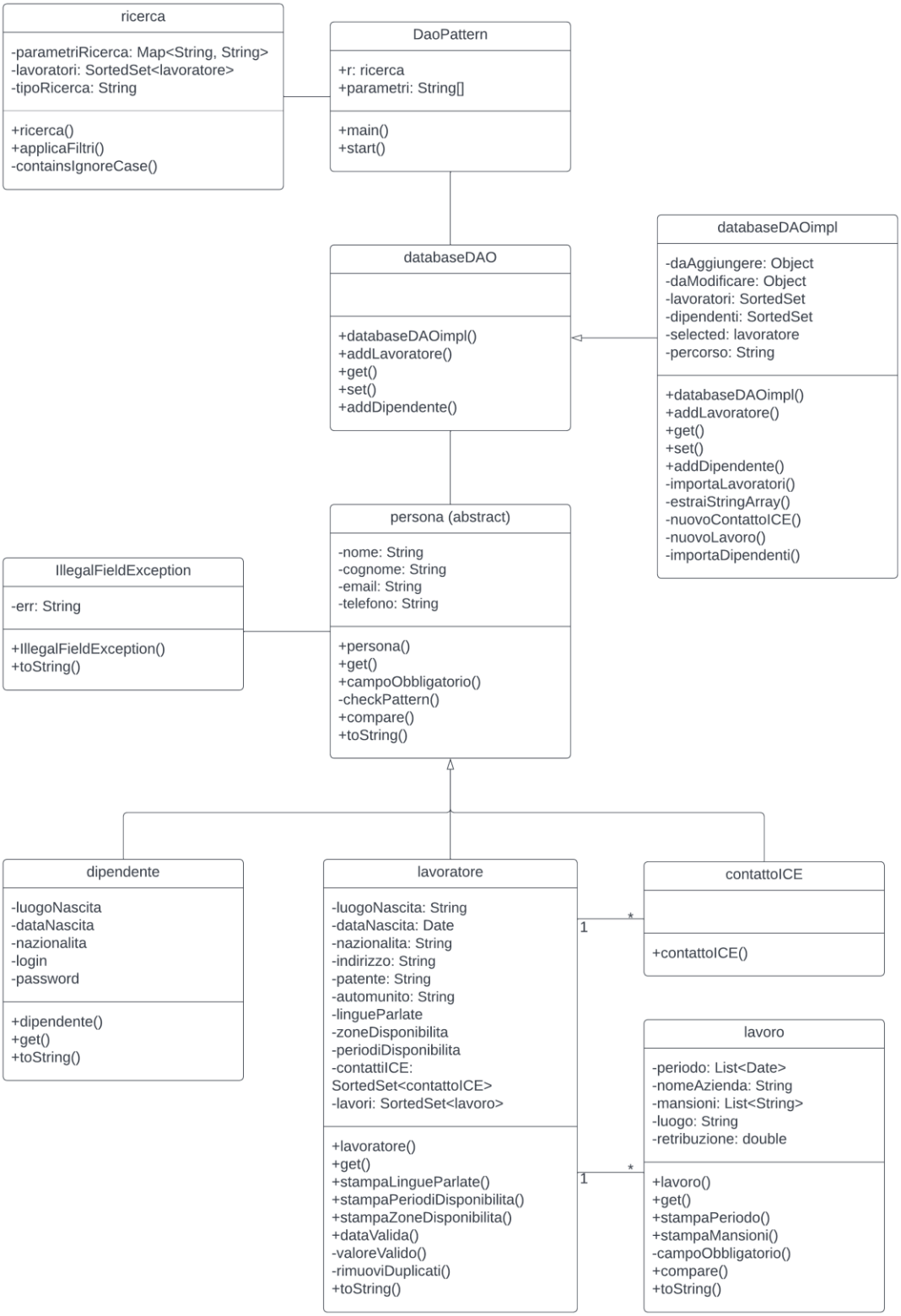
2. **databaseDAOimpl**: questa classe definisce i metodi dichiarati nell'interfaccia (tutti dichiarati come *public*) per permettere al sistema di accedere ai dati, ma definisce anche altri metodi dichiarati *private* usati solo all'interno della classe come *importaLavoratori*() e *importaDipendenti*().

3. **DaoPattern**: questa classe (che comprende anche il metodo *main*()) si occupa di dichiarare un istanza di *databaseDAOimpl* permettendo a tutte le classi di vedere i dati legati a quest'ultimo oggetto (ovvero tutti i dati di lavoratori e dipendenti).



(Figura) Diagramma pattern DAO

Package Model



Le classi del package Model sono state organizzate secondo il diagramma UML sopra per una gestione efficace dei dati. Sono analizzate solo le interazioni salienti.

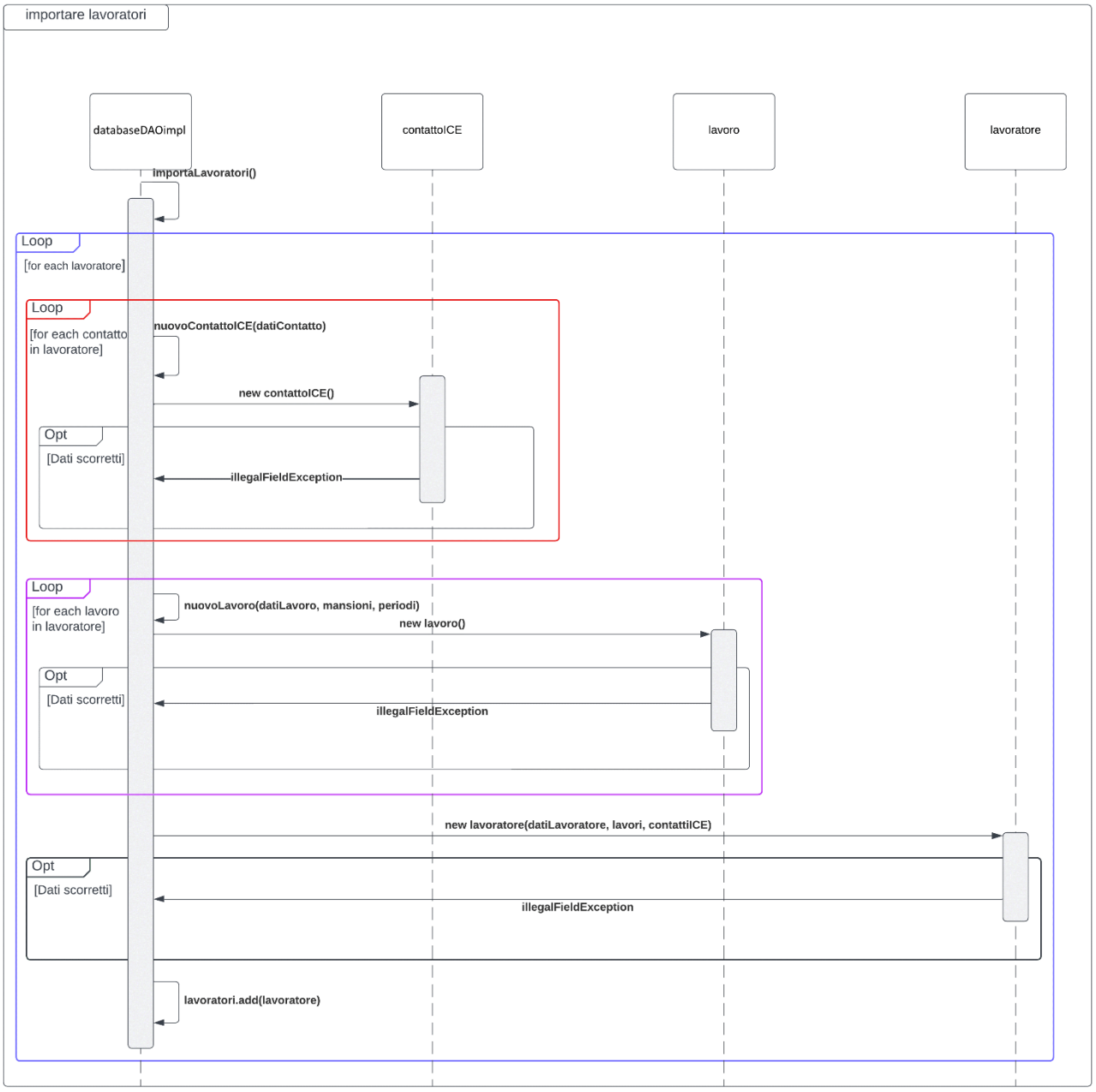
Classe DaoPattern
Questa classe estende Application e avvia la grafica JavaFX tramite il metodo *launch*() chiamato nel *main*(). Prima di avviare la grafica però viene dichiarato e inizializzato l'oggetto database:

databaseDAO database = new databaseDAOimpl()

Classe databaseDAOimpl
Questa classe importa i dati di lavoratori e dipendenti da un file esterno scritto in JSON e li memorizza. Il supporto a JSON è stato realizzato grazie alla libreria JSON.simple (Documentazione: https://code.google.com/archive/p/json-simple/).
Si fa notare che l'esistenza di un database JSON non sarebbe accettabile in una situazione reale (in quanto credenziali e dati sarebbero visibili a tutti in chiaro) e che tale è stato sviluppato solo per facilitare le attività di test. In questo modo infatti ogni attività di test ha avuto disponibile fin da subito un discreto numero di lavoratori su cui testare le singole funzionalità senza bisogno di inserirne di nuovi a ogni avvio dell'applicazione. In una situazione reale tali dati sarebbero stati salvati in un database remoto non visibile all'utente finale, tale non è stato implementato poiché non affrontato durante il corso.

Una volta importati i dati dal database vengono salvati localmente e ogni operazione su questi non modifica il database originale.

Più nel dettaglio, la funzione *importaLavoratori()* accede al database JSON per importare i dati all'interno di *for each* eseguito per ogni lavoratore trovato. Durante ogni ciclo vengono importati i dati dei lavoratori più i dati di ogni lavoro e contatto d'emergenza associato al lavoratore con un nuovo *for each* per i lavori e uno per i contatti.
Ogni volta che sono stati importati tutti i dati per un lavoro questo viene istanziato come oggetto della classe *lavoro* tramite la funzione *addLavoro()*.Questa funzione usa i dati estratti dal database per creare un oggetto lavoro tramite il relativo costruttore, impedendo la creazione dell'oggetto se viene riscontrato un errore. Se invece l'oggetto è valido il lavoro appena creato viene aggiunto al *SortedSet<lavoro> lavori* e si passa alla creazione del lavoro successivo.
Per i contatti d'emergenza viene eseguito un processo analogo al termine del quale tutti i contatti sono salvati all'interno di un *SortedSet<contattoICE> contatti*.
Una volta importati lavori e contatti, alla fine del ciclo di *importaLavoratori()* eseguito per ogni lavoratore viene creato un nuovo oggetto lavoratore passando i dati anagrafici, i lavori e i contatti. Se il lavoratore non è valido viene scartato, altrimenti viene aggiunto al *SortedSet<lavoratore> lavoratori*.



Classe ricerca

Ad ogni nuova ricerca effettuata dall'utente viene istanziato un nuovo oggetto della classe ricerca al fine di mantenere i parametri specificati anche cambiando schermata. Ogni volta che l'utente specifica dei nuovi parametri per la ricerca l'elenco dei lavoratori non visualizzerà il set completo dei lavoratori ma solo quelli restituiti dalla funzione *applicaFiltri*() per i parametri impostati dall'utente. Se i parametri impostati dall'utente non sono validi (si veda la sezione Controlli implementati), verrà visualizzato un messaggio di errore.

Più nel dettaglio, ogni volta che si accede alla schermata con l'elenco dei nuovi lavoratori il sistema

verifica se esiste un istanza della classe ricerca o se questa è uguale a null. Se la trova, invece di visualizzare il risultato di:

database.getLavoratori()

Saranno visualizzati i lavoratori restituiti dalla funzione:

ricerca.applicaFiltri()

Quando vengono definiti dei parametri di ricerca dall'utente infatti il sistema li memorizza all'interno di una nuova istanza della classe ricerca, quindi ricarica la pagina *elencoLavoratori*. In fase di inizializzazione viene verificato se questi parametri esistono.

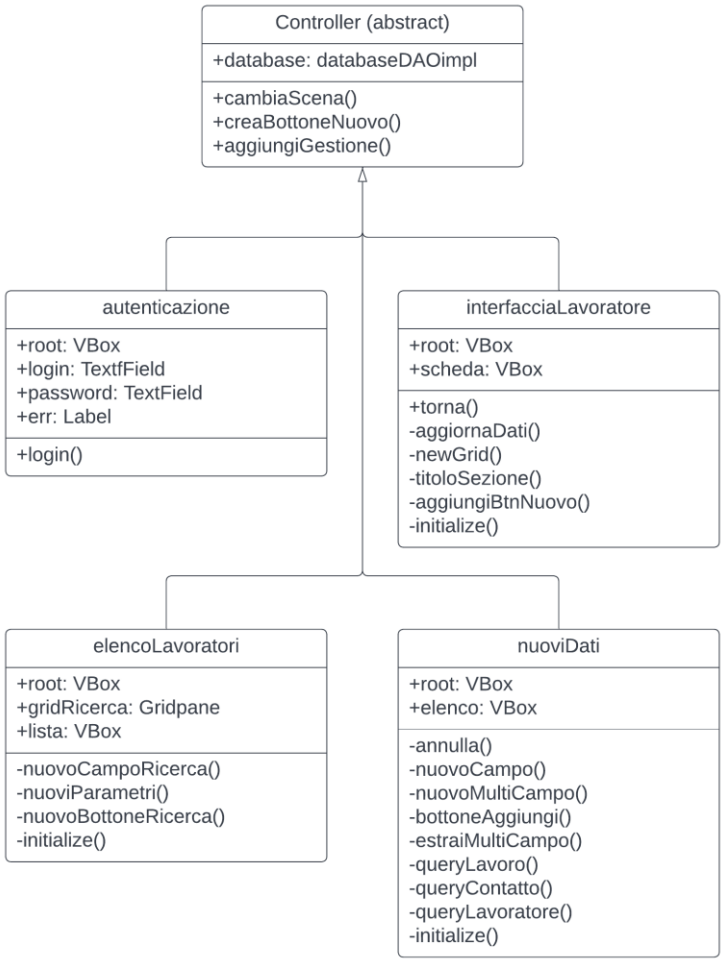
Classe persona
Attributi e metodi in comune alle classi lavoratore, dipendente e contattoICE (In Case of Emergency) sono stati inseriti nella classe astratta persona da cui tutti e tre ereditano. Dopo la chiamata del costruttore padre tutte e tre le classi aggiungono nel proprio costruttore i propri controlli specifici. Per esempio, persona non richiede che il campo telefono sia obbligatorio poiché lavoratore non lo richiede, perciò dipendente e contattoICE implementano un controllo aggiuntivo per verificare non sia assente.
In caso di errore le classi lanciano l'errore personalizzato IllegalFieldException(), a cui viene passata la stringa che specifica l'errore e che verrà stampata all'occorrenza per informare l'utente.

Package Controller

Le classi del package Controller sono state organizzate secondo il diagramma UML sopra allo scopo di permettere un efficace e completo controllo sui dati. Sono analizzate solo le interazioni salienti.

Classe Controller

La classe astratta Controller contiene alcuni metodi generici usati dalle altre classi (che ereditano tutte da Controller) come la funzione *cambiaScena*() che permette di cambiare schermata mantenendo le dimensioni attuali della finestra o la funzione *aggiungiGestione*() che aggiunge i tasti per la modifica e la cancellazione per un lavoratore/lavoro/contatto (usata da *elencoLavoratori* e *interfacciaLavoratore*)



Classe interfacciaLavoratore

Questa classe si occupa di visualizzare la vista dettagliata del lavoratore selezionato, aggiungendo all'interno di un ciclo la creazione della grafica "dinamica". Ad esempio, per ogni lavoro e per ogni contattoICE che risulta collegato al lavoratore selezionato viene creato un riquadro per la loro visualizzazione e i bottoni utili alla loro gestione (modifica lavoro, elimina lavoro, nuovo lavoro).

A questa schermata si viene indirizzati anche dopo aver inserito un nuovo lavoratore in modo da poter modificare i dati inseriti o aggiungere quelli opzionali (altri lavori o nuovi contatti d'emergenza)

Classe nuoviDati

Questa classe permette simultaneamente la modifica o l'aggiunta di un nuovo lavoratore/lavoro/contattoICE in base all'interazione dell'utente. Per uno sviluppo incrementale si è scelto di iniziare con l'implementazione della funzione per aggiungere un nuovo contatto, estendendo poi la funzionalità anche alla modifica dello stesso (possibile semplicemente inizializzando i *TextField* con i valori del contatto selezionato per la modifica).

A questo punto si è aggiunta per entrambe le funzioni il supporto a un tipo di dato generico, permettendo di identificare tramite la funzione *instanceof* quale tipo di oggetto era richiesto modificare o aggiungere tra lavoro e contattoICE.

Come ultimo passaggio si è generalizzata ancor di più la funzione per accettare anche la modifica delle anagrafiche del lavoratore e l'aggiunta di un nuovo lavoratore. Quest'ultima funzione si appoggia sulla modifica di un lavoratore esistente (ma senza passaggio di parametri) e una chiamata iterativa della stessa classe *nuoviDati* chiedendo l'aggiunta di un nuovo contattoICE (poiché ogni lavoratore ne richiede almeno uno). Specificato anche il contatto d'emergenza si viene indirizzati alla vista dettagliata del nuovo lavoratore dov'è possibile eventualmente aggiungere un nuovo lavoro secondo le funzionalità già definite nella classe *interfacciaLavoratore*.

Controlli implementati

I controlli per le quattro classi instanziabili del progetto (dipendente, lavoratore, lavoro e contattoICE) sono stati implementati all'interno dei relativi costruttori e del costruttore della classe astratta persona. Qui l'elenco dei controlli effettuati sugli input forniti dall'utente.

Campi obbligatori

Per tutti i campi obbligatori (segnalati dal sistema con un *) è richiesto che il campo non venga lasciato vuoto.

Email e numero di telefono validi

I controlli su questi attributi sono verificati dalla classe persona che tramite la funzione *checkPattern()* verifica che questi siano validi.

Date generiche

Per ogni data inserita viene effettuato un controllo preventivo che siano date valide scritte nel formato dd/MM/yyyy. In base al tipo di campo in cui sono richieste sono implementati poi controlli più specifici.

Date di nascita

Per le date di nascita di dipendenti e lavoratori si è verificato che esse non siano precedenti al 01/01/1900, valore non verosimile per una data di nascita di un lavoratore/dipendente, e che non siano successive alla data odierna.

Periodi di disponibilità

Su i periodi di disponibilità di un lavoratore sono stati effettuati vari controlli:

1. Viene verificato che la data sia in un formato valido
2. Verifica che sia specificata un inizio e una fine
3. Verifica che l'inizio non sia successivo alla fine
4. Verifica che la data di fine non sia una data inverosimile (successivo al 01/01/2100)
5. Verifica che non ci siano ripetizioni tra i vari periodi di disponibilità, nel caso li ignora

Set di valori predefiniti

Per i campi automunito e patente sono disponibili solo alcuni valori. Viene controllato se il valore inserito dall'utente corrisponde a quelli permessi ignorando maiuscolo/minuscolo. I valori validi sono:

- Per automunito: [Si, No]
- Per patente: [No, M, A, B1, B, C1, C, D1, D, BE, C1E, CE, D1E, DE, T, F]

Valori duplicati

Per i campi *linguеParlate* e *zoneDisponibilita* vengono rimossi i valori duplicati.

Retribuzione valida

Viene controllato che il campo retribuzione di un lavoro sia un numero valido.

Periodo di lavoro

Viene verificato che il periodo di un'esperienza precedente risalga agli ultimi 5 anni come richiesto dal cliente. Viene inoltre controllato che la fine di tale periodo non sia nel futuro.

Attività di test

Per verificare la correttezza dell'applicazione sviluppata sono state svolte le seguenti attività di validazione:

1. Verifica della coerenza tra diagrammi di classe prodotti in fase di progettazione e requisiti specificati dalla consegna.
2. Verifica tra i diagrammi di classe il software prodotto.
3. Test dello sviluppatore sul sistema a ogni nuova funzionalità implementata.
4. Attività di refactoring.
5. Attività di test tramite peer-review.

Test dello sviluppatore

I test sulle funzionalità effettuati dal sottoscritto si sono svolti innanzitutto allo scopo di verificare il corretto funzionamento del software in base a ciò che era il comportamento atteso. Verificato che il software rispondesse correttamente si è quindi valutato il comportamento dell'applicazione di fronte a un uso scorretto dello stesso, verificando che il software potesse reagire di fronte a un utilizzo scorretto da parte di un utente e potesse informarlo in modo chiaro su eventuali errori.

I test effettuati hanno compreso, ma non si sono limitati a, test sulle funzionalità anche in situazioni anomale a scopo di testare la robustezza del software, come ad esempio in assenza di un database pre-esistente e testando le funzionalità solo su lavoratori inseriti manualmente.

Per quanto riguarda i test delle funzionalità, ogni funzione è stata testata numerose volte sia in situazioni normali sia con input scorretti sia prima che dopo l'attività di refactoring. Ci si è anche accertati che tali funzioni fossero affidabili in ogni situazione e che un particolare ordine delle azioni non potesse portare a situazioni di errore.

Test tramite peer-review

Come ultima fase di verifica si è inviato il software eseguibile a dei colleghi in materia perché ne verificassero il funzionamento, portando alla scoperta di ulteriori problemi, segnalati e poi risolti. I problemi riscontrati in questa fase sono stati principalmente sulla mancanza di controlli adeguati sull'input dell'utente, mentre User Interface e funzionalità non hanno subito ulteriori modifiche.