# SeasonalWorkApp


*Davide Bragantini*
*VR455961*
*Michele Franchini*
*VR457435*
*Alessio Zattoni*
*VR457153*

UNIVERSITÀ
di VERONA

*July 2022*

# Indice

# UML

In questa sezione sono presenti i seguenti diagrammi uml che descrivono il sistema:

- Diagramma dei casi d'uso
- Diagrammi di sequenza
- Diagramma di sequenza del sistema implementato
- Diagrammi delle attività
- Diagramma delle classi

## 1.1 Use case diagram

Il sistema software prevede che un dipendente possa, sempre previa autentificazione, inserire ed aggiornare i dati dei lavoratori, inoltre potrà anche ricercare i profili adatti ai relativi lavori stagionali.

### 1.1.1 Use case

In questa sezione viene mostrato il diagramma dei casi d'uso relativi al sistemi in oggetto. Sono stati indentificati 4 casi d'uso:

1. **Insert Record:** *Il dipendente può inserire dati dei lavoratori nel sistema*
2. **Login:** *Il dipendente deve loggarsi per accedere al sistema*
3. **Search:** *Il dipendente può cercare profili adatti ai lavori disponibili*
4. **Update Records:** *Il dipendente può aggiornare i dati dei lavoratori*

Inoltre è stato identificato un unico attore chiamato **Employer**.

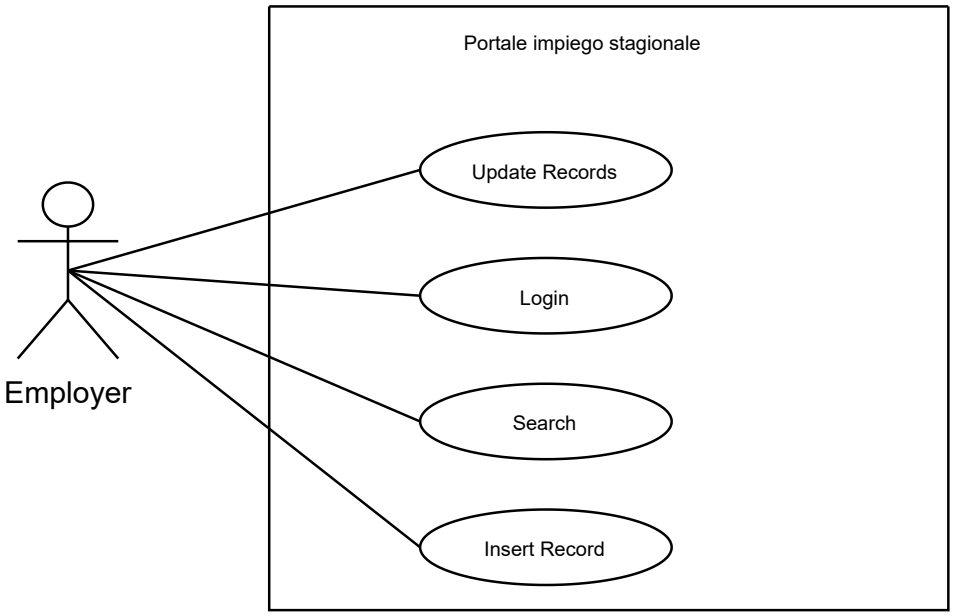


Figura 1.1: Diagramma casi d'uso del sistema

### 1.1.2 Use case specification

In questa sezione vengono definiti e specificati i vincoli e i passi di esecuzione dei vari casi d'uso individuati nello schema precedente.

- ***Search***

***Caso d'uso:*** *Search*

***Id:*** *UC2*

***Attori:*** *Employer*

***Precondizioni:***

- *Il dipendente deve essersi autenticato*
- *Il record relativo deve esistere*

***Passi:***

1. *L'impiegato accede al sistema*
2. *L'impiegato ricerca rispetto ai profili richiesti*

***Postcondizioni:*** *nessuna*

- ***Insert Record***

  ***Caso d'uso:*** *Insert Record*

  ***Id:*** *UC1*

  ***Attori:*** *Employer*

  ***Precondizioni:*** *Il dipendente deve essersi autenticato*

  ***Passi:***

  1. *L'impiegato accede al sistema*
  2. *L'impiegato inserisce i dati del lavoratore*
  3. *Se i dati sono stati inseriti correttamente*
     (a) *le informazioni vengono salvate nel sistema*
  4. *Altrimenti viene mostrato un messaggio per indicare il tipo di errore*

  ***Postcondizioni:*** *I dati del lavoratore sono presenti nel sistema*

- ***Login***

  ***Caso d'uso:*** *Login*

  ***Id:*** *UC3*

  ***Attori:*** *Employer*

  ***Precondizioni:*** *Il dipendente deve essere registrato nel sistema*

  ***Passi:***

  1. *Il dipendente inserisce le credenziali nel relativo form di login*
  2. *Se queste sono corrette*
     (a) *L'impiegato accede al sistema*
  3. *Altrimenti deve ripetere la procedura*

  ***Postcondizioni:*** *L'utente ha accesso al sistema*

- ***Update Records***

***Caso d'uso:*** *Update Records*

***Id:*** *UC4*

***Attori:*** *Employer*

***Precondizioni:***

- *Il dipendente deve essersi autenticato*
- *Il record relativo deve esistere*

***Passi:***

1. *L'impiegato accede al sistema*
2. *L'impiegato modifica i dati del lavoratore*
3. *Se i dati sono stati inseriti correttamente*
   (a) *I dati del lavoratore vengono salvati nel sistema*
4. *altrimenti viene mostrato un messaggio per indicare il tipo di errore*

***Postcondizioni:*** *I dati del lavoratore sono stati aggiornati*

## 1.2 Sequence Diagram

In questa parte vengono presentati i diagrammi di sequenza UML. Per ogni caso d'uso è stato prodotto un diagramma che enfatizza lo scambio di messaggi nel tempo.

### 1.2.1 Sequence Diagram

**LOGIN**

L'impiegato inserirà le credenziali nell'apposito form di login, il sistema verificherà se esistono e se sono corrette, se le credenziali sono corrette l'impiegato accederà al sistema, altrimenti dovrà riprovare l'inserimento delle credenziali.

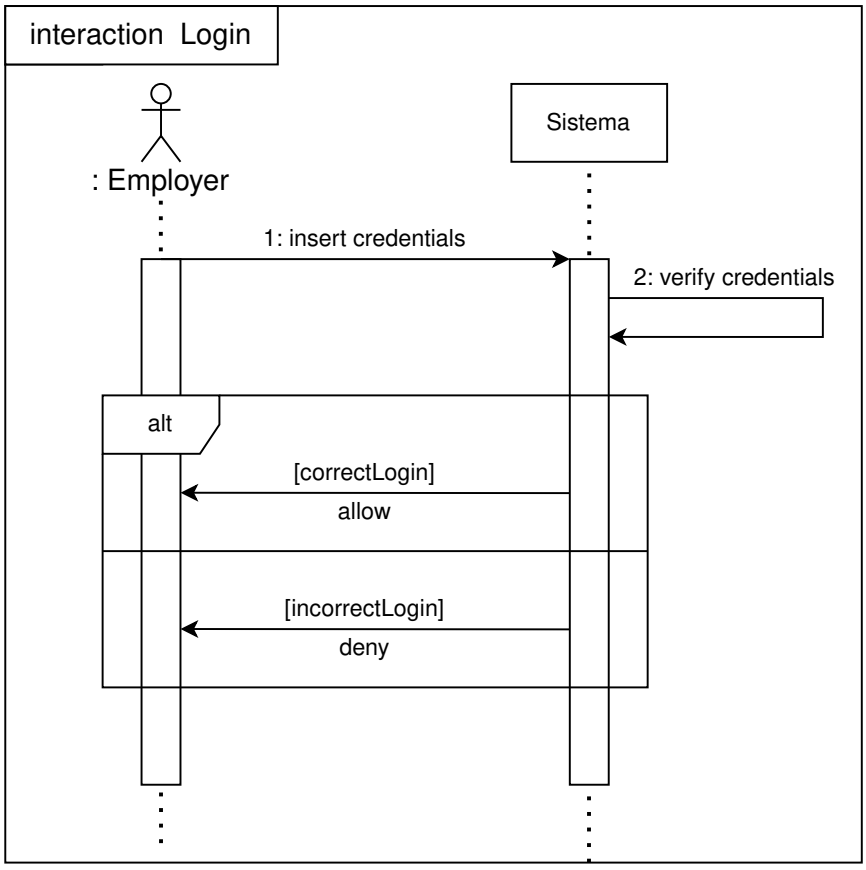


Figura 1.2: Login sequence diagram

### INSERT RECORD

Il dipendente proverà ad inserire i dati relativi ad un nuovo lavoratore stagionale, il sistema verificherà se i dati sono ammisibili, se i dati inseriti sono accettabili, allora i dati del lavoratore verranno salvati nel sistema, altrimenti l'impiegato dovrà reinserire i dati nell'apposito form.

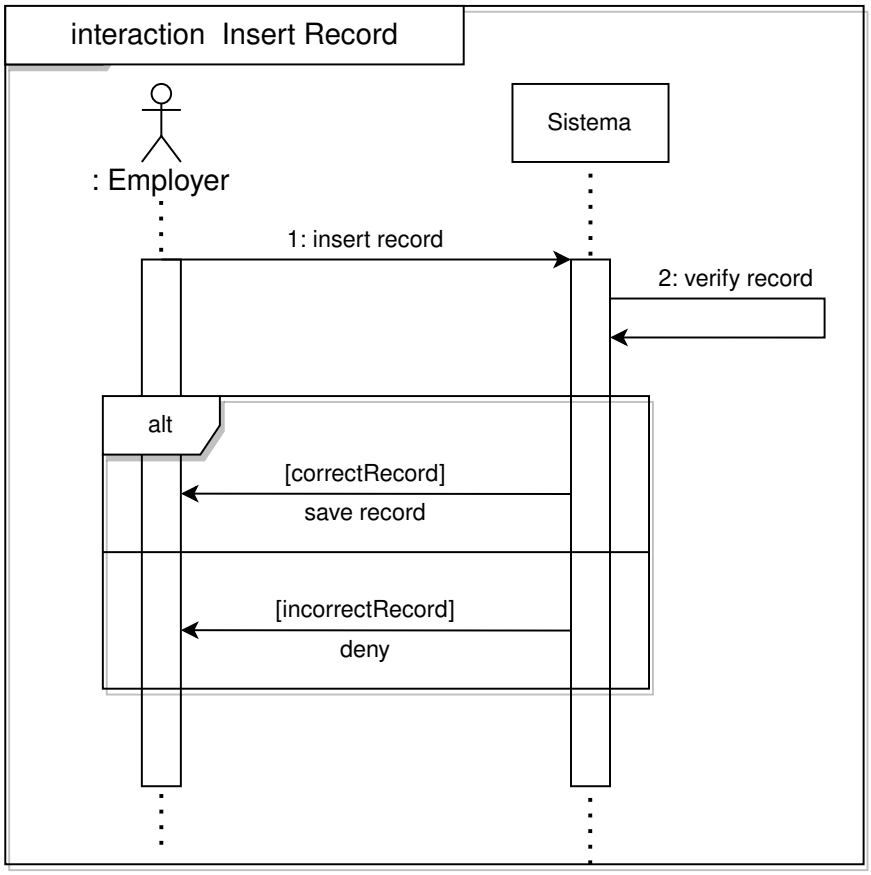


Figura 1.3: Insert Record sequence diagram

### SEARCH

Il dipendente potrà ricercare i profili adatti ai diversi lavori stagionali compilando un form di ricerca, se la ricerca va a buon fine allora verranno mostrati i lavoratori compatibili con i filtri impostati, altrimenti non verrà mostrato nessun lavoratore.

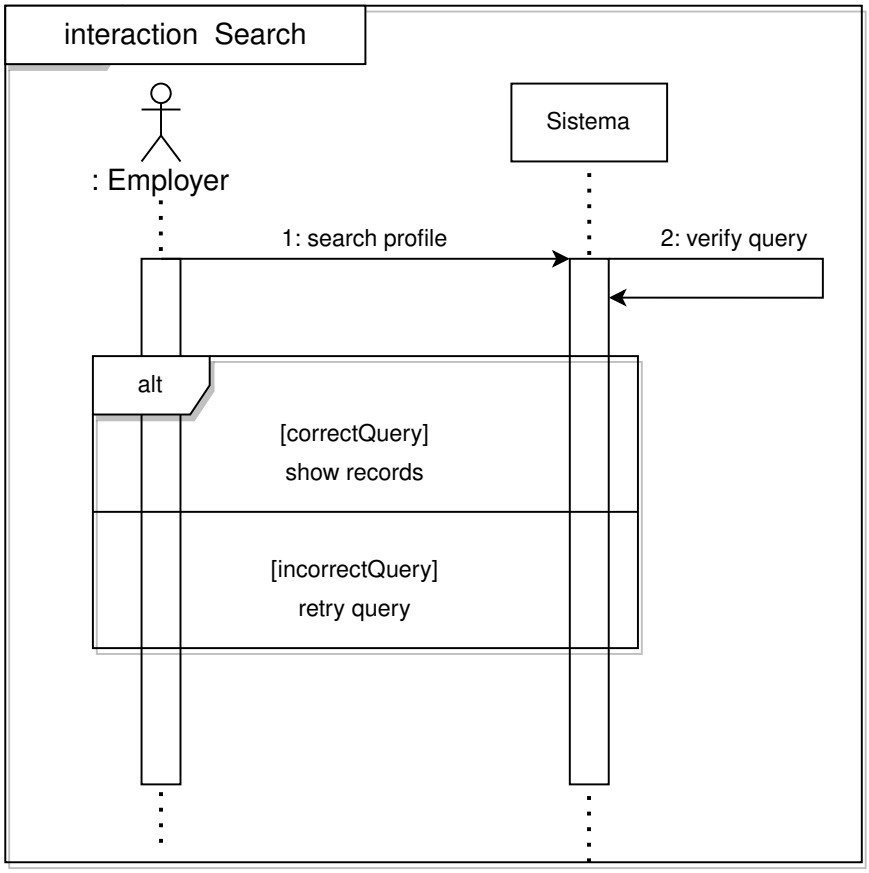


Figura 1.4: Search sequence diagram

## UPDATE RECORD

L'impiegato proverà ad aggiornare i dati di un lavoratore presente nel sistema tramite un form apposito per l'aggiornamento dei dati, il sistema verificherà se i nuovi dati sono coretti, se le informazioni inserite sono accettabili allora le informazioni del lavoratore verranno aggiornate con successo, altrimenti il dipendente dovrà riprovare ad inserire i nuovi dati del lavoratore.

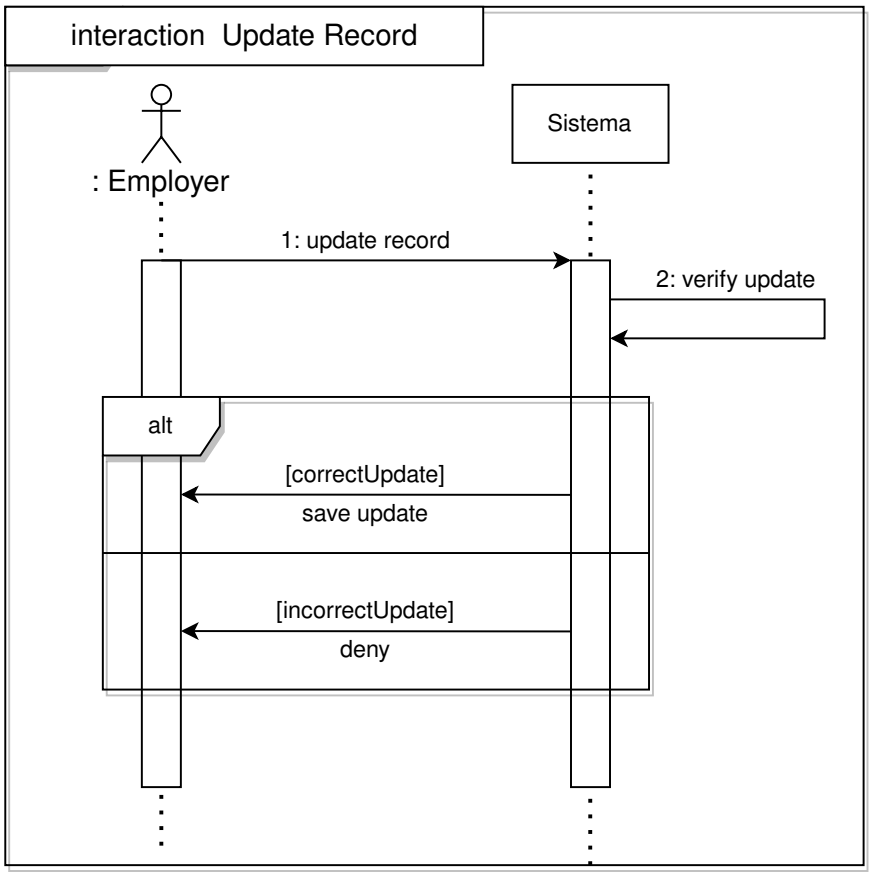


Figura 1.5: Update Record sequence diagram

### 1.2.2 Implemented system Sequence diagram

**LOGIN**

Il dipendente inserisce le credenziali nell'interfaccia *Login.fxml*, dopo aver premuto login il controllore *LoginController* processa la richiesta inviandola al DAO tramite il metodo *Login*, il DAO controlla se le credenziali sono corrette. Se le credenziali sono corrette l'utente viene indirizzato alla vista *Home.fxml*, altrimenti verrà mostrato un messaggio di errore nell'interfaccia *Login.fxml*.

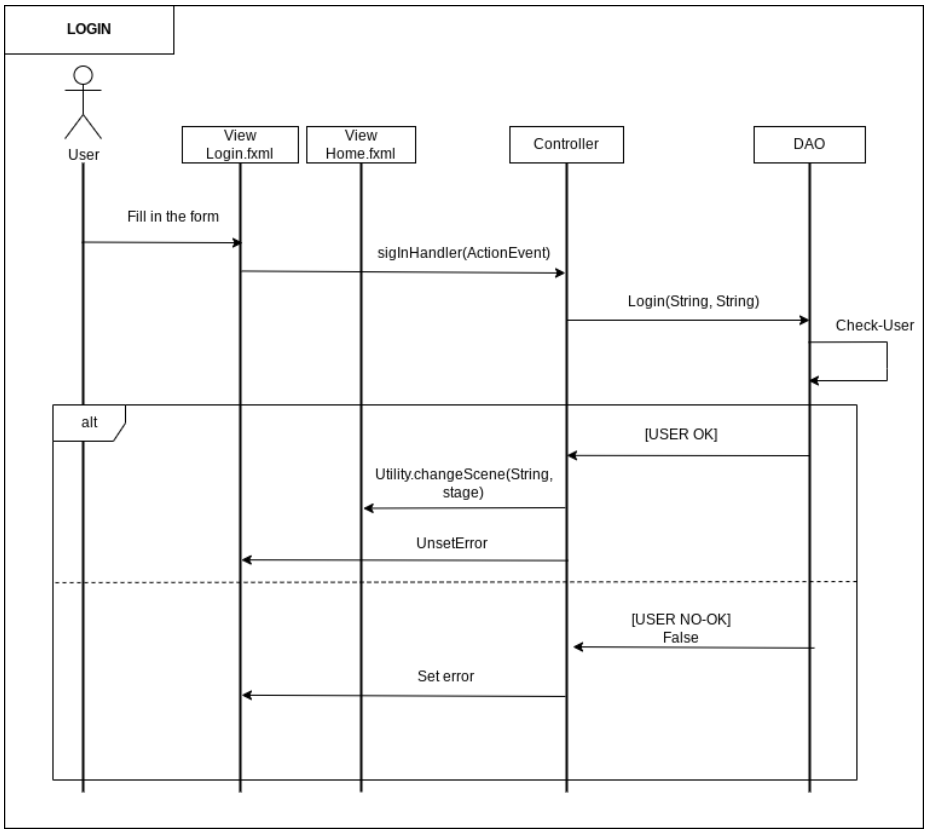


Figura 1.6: Login sequence diagram

### INSERT RECORD

L'utente potrà inserire i dati del lavoratore nell'interfaccia *Insert.fxml*, una volta premuto next il controllore *InsertController* processa i dati, se i dati non sono corretti allora viene stampato un messaggio di errore altrimenti l'utente viene indirizzato all'interfaccia *InsertExp.fxml*, in questa vista l'utente potrà inserire i dati relativi alle esperienze passate del lavoratore e successivamente salvare il lavoratore. Il controllore *InsertExpController* verificherà la correttezza delle esperienze passate, tramite il metodo *addExpHandler*, se tutto è corretto allora verrà inserita l'esperienza passata altrimenti verrà stampato un messaggio di errore. Il tasto submit permette di inserire il record nel database, infatti una volta premuto tramite il metodo *submitHandler* verrà richiamato il servizio *addRecord* della classe DAO, questo metodo scriverà il record nel database, una volta finito il controller indirizzato l'utente in *Home.fxml*.

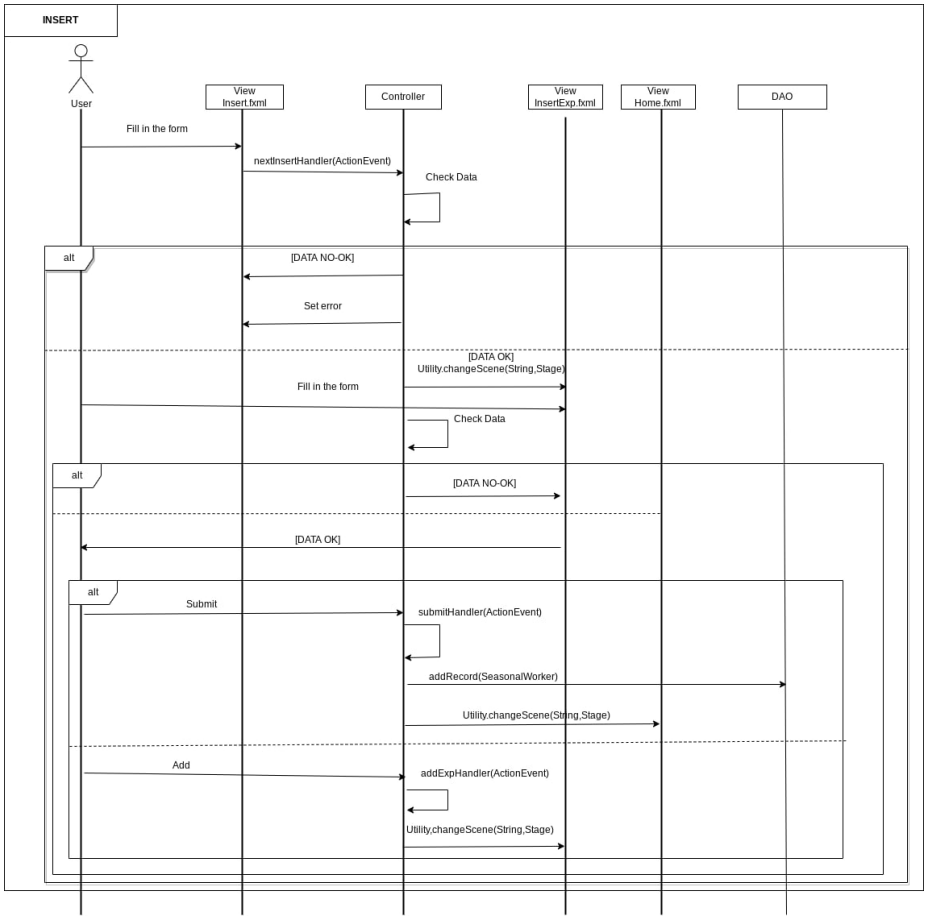


Figura 1.7: Insert sequence diagram

## UPDATE RECORD

Una volta premuto il pulsante update in una delle viste, il controllore indirizzerà l'utente in *UpdateChoice.fxml*, una volta qui il dipendente dovrà scegliere il lavoratore da aggiornare tramite id, se il lavoratore non ha l'id con se si potrà utilizzare il servizio di ricerca. Una volta premuto next il controllore *updateChoiceController* tramite il metodo *nextUpdateHandler* porterà l'utente alla vista *Update.fxml*. In questa vista il dipendete potrà non obligatoriamente riempire il form, se i dati sono corretti *UpdateController* porterà l'utente alla vista *UpdateExp*, altrimenti stamperà un messaggio di errore nell'interfaccia. Una volta arrivati all'interfaccia *UpdateExp.fxml*, l'utente potrà inserire le esperienze passate, una volta inserite il controllore *UpdateExpController* verificherà la correttezza delle esperienze passate, tramite il metodo *updateExpHandler*, se tutto è corretto allora verrà aggiornata l'esperienza passata altrimenti verrà stampato un messaggio di errore. Il tasto update permette di aggiornare il record nel database, infatti una volta premuto tramite il metodo *updateHandler* verrà richiamato il servizio *updateRecord* della classe DAO, questo metodo aggiornerà il record nel database, una volta finito il controller indirizzato l'utente in *Home.fxml*.

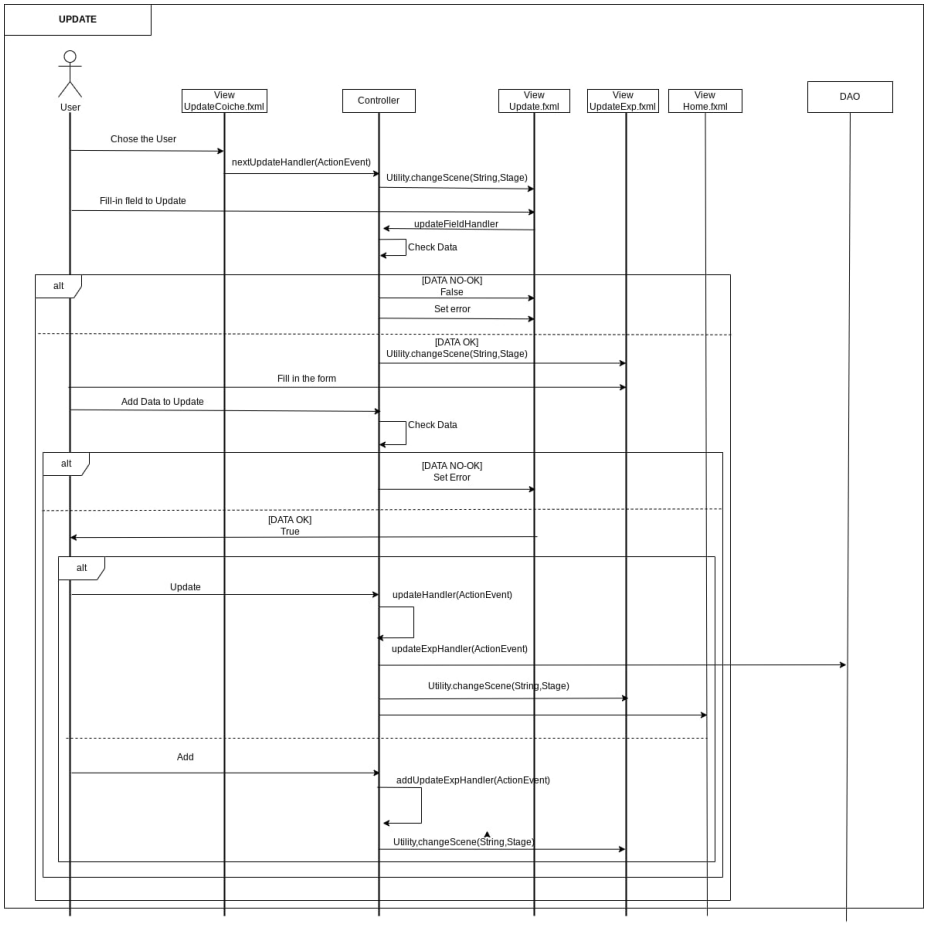


Figura 1.8: Update sequence diagram

## SEARCH RECORD

Una volta premuto il pulsante *search* in una delle viste, l'utente verrà portato alla vista *Search.fxml* dove potrà ricercare i lavoratori tramite filtri. Una volta qui il dipendente potrà appunto selezionare dei filtri per filtrare i lavoratori, una volta premuto il pulsante *search* il controller *SearchController* eseguirà una ricerca per vedere se ci sono record che rispettano i filtri, se non ci sono allora non verrà mostrato nulla altrimenti la tabella verrà aggiornata con i nuovi dati, una volta disponibili i vari record l'utente può selezionare un record e tramite il doppio click, verrà invocato l'handler che gestisce l'evento. Questo handler porterà l'utente alla vista *SearchResult.fxml* dove potrà ispezionare i dati del lavoratore interessato.

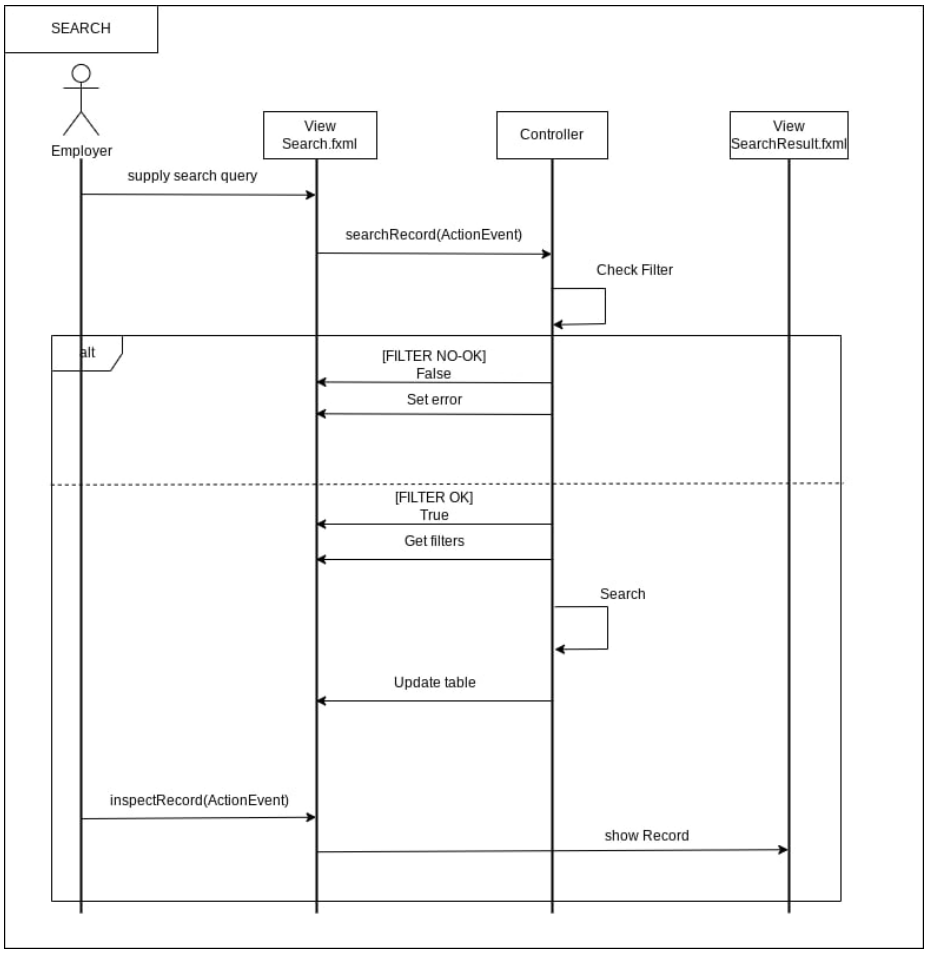


Figura 1.9: Search sequence diagram

## 1.3 Activity diagram

In questa parte verranno mostrati i diagrammi delle attività relativi all'intero sistema, inoltre verrà mostrato il diagramma per la fase di login.

### 1.3.1 Activity diagram

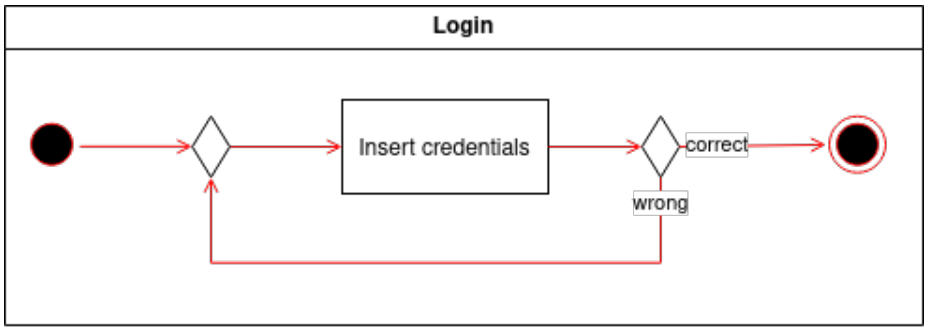


Figura 1.10: Login activity diagram

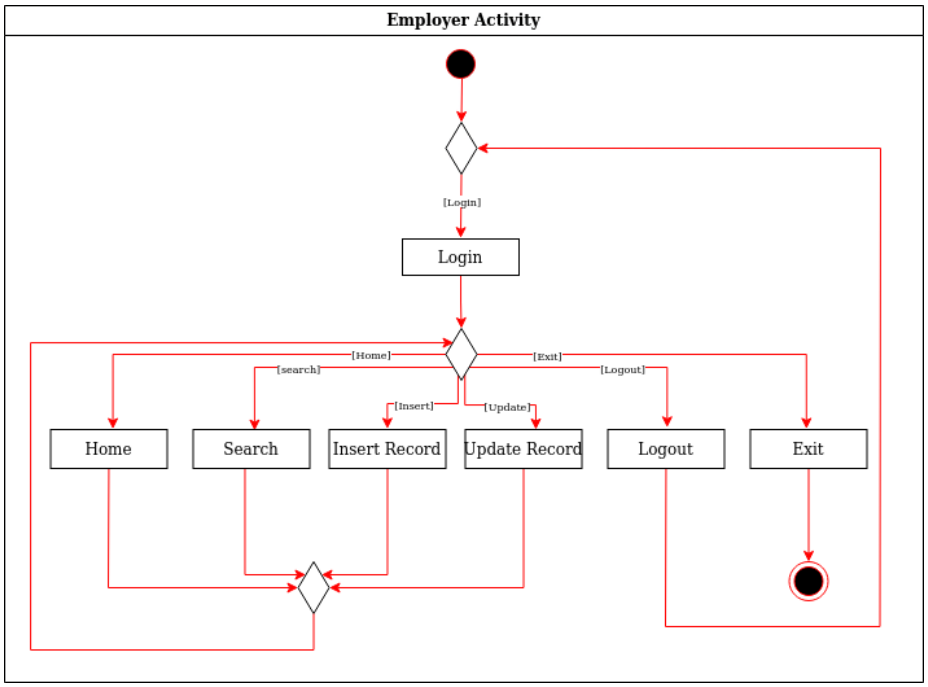


Figura 1.11: Employer activity diagram

## 1.4 Class Diagram

In questa parte verrà illustrata la struttura delle classi del progetto con le loro relazioni. Questo diagramma permette di avere una vista più vicina al codice, inoltre mostra le relazioni tra le varie classi.

### 1.4.1 Class Diagram

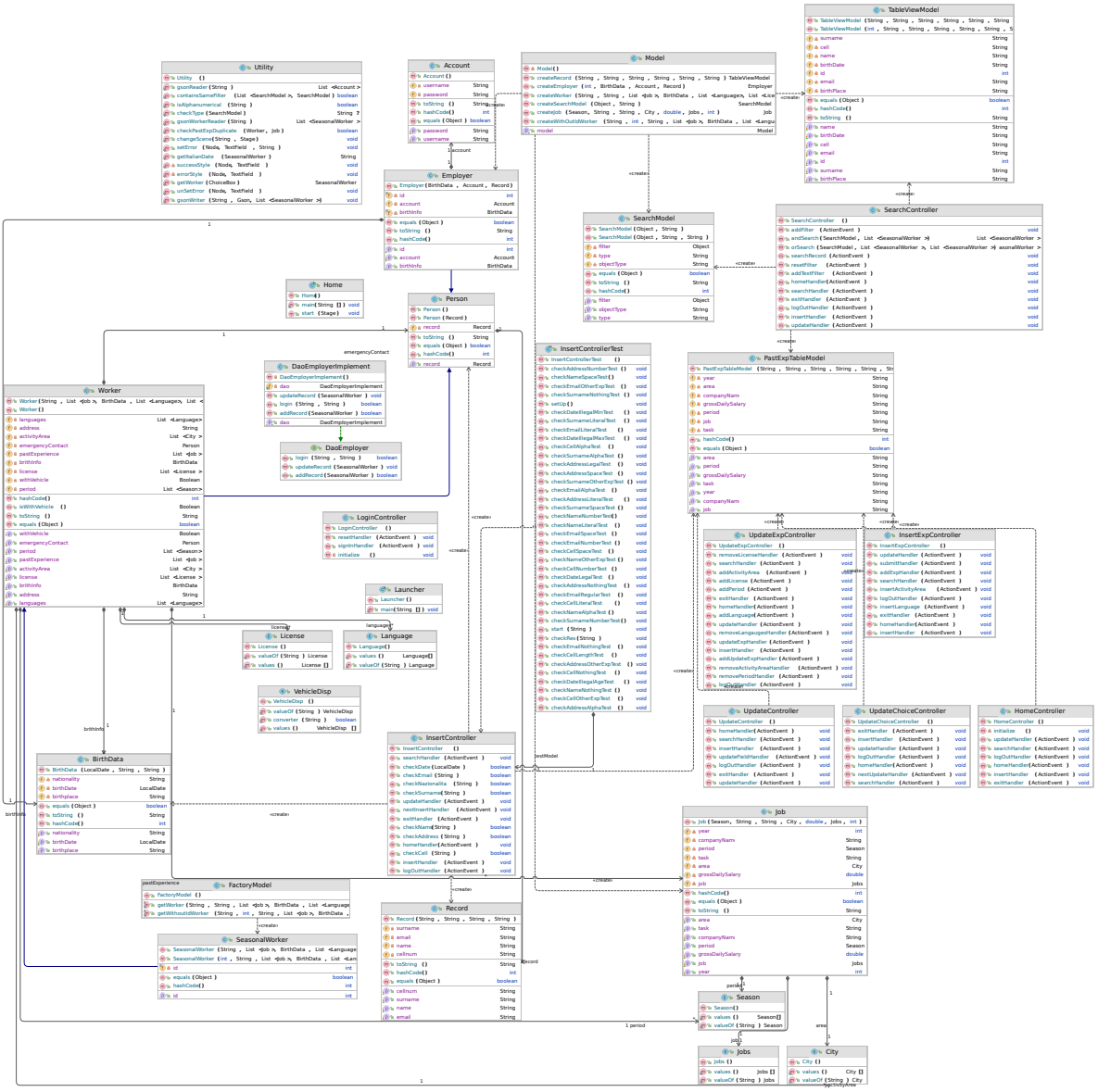

Figura 1.12: System class diagram

# Project pattern

## 2.1 Architectural patterns

In questa sezione analizzeremo il pattern architetturale utilizzato per la realizzazione del software. Il modello scelto per il sistema software caratterizza il software stesso e ne definisce la struttura.

### 2.1.1 MVC

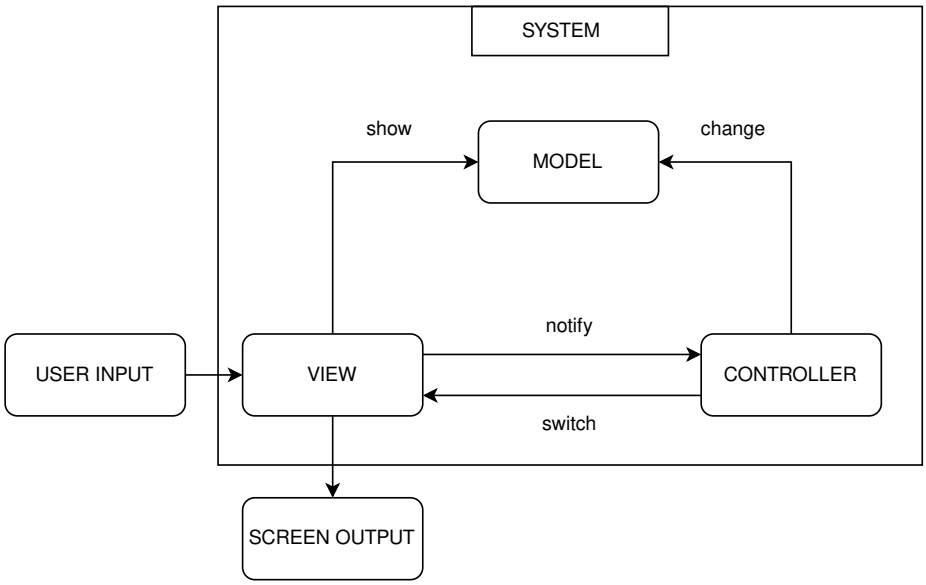


Figura 2.1: Architectural pattern diagram - MVC

Il sistema si basa sul modello MVC, costituito da:

1. ***Model***: rappresenta il sistema.
2. ***View***: ci permette di rappresentare il modello con varie interfacce.
3. ***Controller***: agisce sul modello modificandone lo stato effettuando delle operazioni.

Il *Model* è rappresentato da varie classi presenti nel package *Model*, su di esso agisce il *Controller* che rappresenta la logica applicativa del sistema. Esso è implementato tramite **listeners** che alterano lo stato del modello. La *View* infine viene implemetata tramite fxml, così da renderla indipendente rispetto al resto dei componenti ed essa varierà grazie agli eventi. Questo modello serve per separare i tre **layer**, si presta bene in questa architettura dove vi sono varie viste "caricate" dal controllore senza la necessita di modificare la vista stessa. La separazione dei tre componenti ha il pregio di aumentare la mantenibilità del codice e permette di effettuare in maniera facilitata un'aggiunta di moduli o funzionalità.

## 2.2 Design Pattern

In questa sezione verranno analizzati i design pattern utilizzati per la realizzazione del sistema software. Essi offrono dei template già testati, funzionanti e applicabili in differenti contesti. In caso di necessità di un incremento del software sarà possibile l'introduzione di ulteriori design pattern per preservarne la mantenibilità e la modularità.

### 2.2.1 Singleton Pattern

Questo pattern viene utilizzato nel Model, permettendoci un risparmio di memoria e una maggiore mantenibilità e leggibilità. Viene utilizzato in accoppiata con il design pattern Factory così d'avere un'unica istanza dell'oggetto della classe.

### 2.2.2 Factory Pattern

Questo pattern offre dei particolari servizi di creazione, è stato pensato per astrarre la creazione della classe *Worker*, anche se è presente solo una sottoclasse ***SeasonalWorker*** essa è stata volutamente implementata con la possibilità, in futuro, di estendere il sistema con ulteriori sottoclassi.

### 2.2.3 Dao Pattern

Questo pattern è utilizzato per separare i metodi di basso livello che permettono l'accesso ai dati (es. i getter e setter), rispetto ai servizi di sistema, ovvero quelli identificati nel casi d'uso. Esso viene implementato tramite a un'interfaccia e una classe che ha il compito di implementarla. Se in futuro si volesse estendere la fornitura di servizi (es. una procedura di eliminazione dei record), sarà necessario integrarli nell'interfaccia e implementarli attraverso i metodi di basso livello.

# Software development

## 3.1 Software development method

In questa parte verrà spiegato e motivato l'utilizzo delle metodologie di sviluppo del software relative al progetto, dato che non esiste una metodologia giusta o sbagliata ma dipende da situazione in situazione, noi abbiamo optato per un approccio ibrido preferendo una partenza più rigorsa(plan driven), per poi cambiare metodologia scegliendone una meno stringente(Agile).

### 3.1.1 Software development method

Inizialmente il software è stato sviluppato seguendo una metodologia *plan-driven*, con il fine di ottenere una documentazione più rigorosa e accurata, infatti la fase di progettazione è stata eseguita con questo approccio producendo subito una documentazione quasi competa. Nella fase di implementazione è stato adottato un framework *agile* che ha permesso di passare dalla fase di implementazione a quella di progettazione in maniera più dinamica, permettendoci di integrare opportunamente la documentazione durante lo sviluppo. A partire da questa fase è stata applicata la metodologia *extreme-programming*, in particolare sono stati applicati i concetti di *pair-programming* (rapportato a tre sviluppatori) e un processo continuo di refactoring. In parallelo a ciò è stato utilizzato il framework *Scrum*, effettuando meeting giornalieri (daily scrum) dove nel tempo stabilito, si discuteva dei problemi riscontrati, stato attuale del progetto ed eventuali sviluppi futuri. Il progetto nel complesso è stato sviluppato da tutto il team insieme, non è stato applicato un sistema che prevede la divisione del progetto poiché avrebbe portato ad inconsistenza e problemi nel progetto.

# Software testing

In questa sezione verranno specificati i test eseguiti sul software, nello specifico verranno mostrati i vari livelli di testing dell'applicazione.

## 4.1 Software testing

### 4.1.1 Unit testing

La fase di testing è stata progettata in prima parte con lo sviluppo di unit testing, tramite la libreria *junit*, questi test sono stati prodotti dopo l'implementazione del sistema, in questa parte è stata comunque utilizzata una metodologia agile che ci ha permesso di passare dalla fase di testing a quella di implementazione per permettere di risolvere i bug riscontrati durante la fase di testing. Questi test si sono concentrati principalmente sul controllo dei dati inseriti nei vari form, cercando di far emergere vari bug che avrebbero portato alla non consistenza dei dati nel database, tutti questi test hanno avuto esito positivo. Sono stati svolti i seguenti test:

1. Abbiamo testato il campo **nome** con:
   (a) stringa letterale
   (b) stringa numerica
   (c) caratteri speciali
   (d) stringa vuota
   (e) spazio
2. Abbiamo testato il campo **cognome** con:
   (a) stringa letterale
   (b) stringa numerica
   (c) caratteri speciali
   (d) stringa vuota
   (e) spazio

3. Abbiamo testato il campo **cellulare** con:
   (a) stringa letterale
   (b) stringa numerica
   (c) caratteri speciali
   (d) stringa vuota
   (e) spazio
   (f) stringa più corta di 10 caratteri
4. Abbiamo testato il campo **email** con:
   (a) stringa letterale
   (b) stringa numerica
   (c) caratteri speciali
   (d) stringa vuota
   (e) spazio
5. Abbiamo testato il campo **data** con:
   (a) data con anno $<$ 1950
   (b) data con anno $>$ anno corrente
   (c) data di nascita lavoratore $<$ 16
   (d) data legale
6. Abbiamo testato il campo **address** con:
   (a) stringa letterale
   (b) stringa numerica
   (c) caratteri speciali
   (d) stringa alfanumerica
   (e) stringa vuota
   (f) spazio

Di seguito viene riportato un esempio di test del metodo checkDate.

```
@Test
@DisplayName("Check legal date")
public void checkDateIllegalAgeTest() {
    start(new Throwable().getStackTrace()[0].getMethodName());
    Assert.assertEquals(testModel.checkDate(
    LocalDate.of(2006, 10, 27)), false);
    checkRes( new Throwable().getStackTrace()[0].getMethodName());
}
```

### 4.1.2 System testing

Successivamente sono stati svolti dei test sul sistema dai componenti del team, questi test sono stati fatti direttamente sull'interfaccia del software con lo scopo di scoprire vari bug non scoperti nella fase di unit testing, il sistema in questo caso viene testato nel complesso. In questa fase sono stati eseguiti vari test:

**LOGIN**

1. Abbiamo provato a premere il pulsante "Reset" varie volte senza dati, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato ad inserire dei dati non consistenti come "" e " ", non sono stati riscontrati malfunzionamenti
3. Abbiamo provato ad inserire username e/o password sbagliati, non sono stati riscontrati malfunzionamenti

**HOME**

1. Abbiamo provato a premere i pulsanti relativi ai vari servizi, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato a fare il logout, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato il pulsante per uscire dall'app (exit), non sono stati riscontrati malfunzionamenti
4. Abbiamo provato a cambiare i dati nel text area, non sono stati riscontrati malfunzionamenti

**INSERT**

1. Abbiamo provato ad inserire dati non consistenti, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato ad inserire un lavoratore duplicato, non sono stati riscontrati malfunzionamenti
4. Abbiamo provato ad inserire un lavoratore con il mezzo senza patente, non sono stati riscontrati malfunzionamenti

### INSERT EXP

1. Abbiamo provato ad inserire dati non consistenti, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato ad aggiungere un'esperienza passata vuota, non sono stati riscontrati malfunzionamenti
4. Abbiamo provato ad inserire lingue parlate duplicate, non sono stati riscontrati malfunzionamenti
5. Abbiamo provato ad inserire lingue parlate vuote, non sono stati riscontrati malfunzionamenti
6. Abbiamo provato ad inserire zone di lavoro vuote, non sono stati riscontrati malfunzionamenti
7. Abbiamo provato a premere "Submit" senza dati non sono stati riscontrati malfunzionamenti

### UPDATE CHOICE

1. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato a premere "Next" senza selezionare un lavoratore, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato a selezionare vari lavoratori per vedere se il record della tabella veniva aggiornato, non sono stati riscontrati malfunzionamenti

### UPDATE

1. Abbiamo provato ad inserire dati non consistenti, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato ad inserire dati già presenti nel sistema relativi ad un altro lavoratore senza successo (email, cellulare), non sono stati riscontrati malfunzionamenti
4. Abbiamo provato a premere "Next" senza cambiare alcuno campo con successo, non sono stati riscontrati malfunzionamenti

### UPDATE EXP

1. Abbiamo provato ad inserire dati non consistenti, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato ad aggiungere un'esperienza passata vuota, non sono stati riscontrati malfunzionamenti
4. Abbiamo provato ad inserire dati duplicati, non sono stati riscontrati malfunzionamenti
5. Abbiamo provato ad inserire dati vuoti, non sono stati riscontrati malfunzionamenti
6. Abbiamo provato a rimuovere un dato non presente, non sono stati riscontrati malfunzionamenti
7. Abbiamo provato a rimuovere un dato presente, non sono stati riscontrati malfunzionamenti
8. Abbiamo provato a premere "Update" senza dati non sono stati riscontrati malfunzionamenti

### SEARCH

1. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti
2. Abbiamo provato ad aggiungere filtri duplicati, non sono stati riscontrati malfunzionamenti
3. Abbiamo provato varie combinazioni di filtri non sono stati riscontrati malfunzionamenti
4. Abbiamo provato a premere "Reset" più volte non sono stati riscontrati malfunzionamenti
5. Abbiamo provato a premere "Add filter" più volte non sono stati riscontrati malfunzionamenti
6. Abbiamo provato a premere "Search" senza dati non sono stati riscontrati malfunzionamenti
7. Abbiamo provato a premere "Inspect" senza selezionare dati non sono stati riscontrati malfunzionamenti

**SEARCH RESULT**

1. Abbiamo provato a premere i servizi offerti dal menù, non sono stati riscontrati malfunzionamenti

2. Abbiamo provato a cambiare i dati nel text field, non sono stati riscontrati malfunzionamenti

### 4.1.3 Beta testing

Questa tipologia di test prevede la distribuzione dell'applicazione ad un numero ridotto di utenti finali, nel nostro caso gli stakeholder sono stati i nostri parenti. L'applicazione è stata provata quotidianamente da questi utenti scelti in modo da mettere in risalto i rimanenti bug che il team di sviluppo non ha riscontrato nelle varie fasi di test.

Bug scoperti:

1. In *Search.fxml* non funzionava il filtro per cognome, questo problema è stato risolto

2. In *InsertExp.fxml* e *UpdateExp.fxml* nel campo "nome dell'azienda", non funzionavano stringhe con i punti es. prova s.r.l. questo problema è stato risolto

3. In *Search.fxml* la ricerca per lavoratore doveva essere case insensitive, in questa fase di testing è stato scoperto invece che era case sensitive, questo problema è stato risolto

4. In *Search.fxml* non funzionava il filtro per job, questo problema è stato risolto

# Software specification

In questa sezione verrà mostrata una guida ai servizi del sistema software, inoltre verranno specificate le varie caratteristiche del sistema software.

## 5.1 Software services

Il software, al suo avvio, presenta al dipendente dell'agenzia una schermata di Login, dove ha l'opportunità di autenticarsi per accedere al sistema. Una volta effettuato l'accesso l'utente si trova nella Home dell'applicazione da cui ha la possibilità di eseguire alcune operazioni.

### 5.1.1 Aggiungere un nuovo lavoratore

Per inserire un nuovo lavoratore stagionale si pigia sul tasto "Insert" nel menù a sinistra e si presenta l'interfaccia di aggiunta dipendente. Nella parte alta dello schermo si visualizzano i dati personali del lavoratore ed è necessario compilare tutti i campi richiesti e nel caso in cui quest'ultimi siano stati inseriti incorrettamente viene segnalato colorando di rosso i bordi del campo in questione. Nella parte bassa della schermata invece sono presenti le informazioni professionali del lavoratore: patenti, periodo di disponibilità per lavorare e se automunito. Per procedere alla prossima schermata premere sul bottone "Next" dove si aggiungono gli altri campi premendo il pulsante "Add" nel caso in cui si voglia aggiungere più di una opzione. L'aggiunta delle esperienze lavorative passate è stata progettata in modo tale da lasciare all'utente l'opportunità di aggiungere 0 o più esperienze lavorative passate, ogni volta che si preme "Add experience" questa viene aggiunta al proprio record. Una volta terminata la compilazione si preme "Submit" per salvare il lavoratore appena inserito al sistema. Al termine dell'operazione si verrà reindirizzati alla Home e contemporaneamente verrà mostrato un messaggio per attestare l'avvenuta memorizzazione dei dati.

### 5.1.2 Modificare i dati di un lavoratore

Per aggiornare i dati di un lavoratore già presente nel sistema si pigia sul tasto "Update" nel menù a sinistra e si presenta un'interfaccia in cui è possibile scegliere l'ID del lavoratore da modificare da un menù a tendina. Una volta scelto l'identificativo viene mostrata un'overview del lavoratore in una tabella e pigiando "Next" si procede alla schermata di modifica dei dati. Analogamente all'Inserimento di un nuovo lavoratore i campi inseriti in maniera non corretta verranno colorati di rosso e sarà necessario correggerli per poter procedere oltre. L'utente può scegliere se modificare un campo assegnandoli un nuovo valore (es. cambiare l'indirizzo di posta elettronica) oppure lasciarli bianchi per indicare che verrà salvato il valore già presente nel sistema (per continuare l'esempio di prima, se viene inserita solo una nuova email sarà aggiornato solo quel campo, mentre tutti gli altri valori rimarranno inalterati). Una volta terminato l'inserimento si pigia sul bottone "Update" e i dati verranno registrati nel sistema, successivamente si verrà reindirizzati alla Home e contemporaneamente verrà mostrato un messaggio per attestare l'esito positivo dell'operazione appena conclusa.

### 5.1.3 Cercare un lavoratore

Per cercare un lavoratore tra quelli presenti nel sistema si arriva alla sezione "Search" tramite l'apposito pulsante nel menù a sinistra e si viene accolti da un interfaccia molto ricca di funzionalità che permette di eseguire ricerche complesse nel database dell'Agenzia. La ricerca di eventuali profili richiesti avviene filtrando i dati dei lavoratori (lingue parlate, patenti possedute, periodo di disponibilità, ecc...) tramite una combinazione di opzioni: "Filter" indica la tipologia di filtro da applicare (es. lingua) a cui si concatena il discriminante effettivo (es. Inglese) e infine si sceglie se aggiungere questo filtro in "AND" oppure in "OR" ad eventuali parametri di ricerca aggiuntivi. Se viene selezionato "AND" il sistema restituirà i lavoratori che rispetteranno tutte le condizioni di ricerca (parlano Inglese e sono automuniti), mentre con "OR" si possono cercare più profili contemporaneamente (ad esempio si ricercano i lavoratori in possesso di patente B "OR" patente C per guidare un autovettura). I filtri vengono aggiunti uno ad uno pigiando sul pulsante "Add" e vengono visualizzati nella tabella centrale, una volta soddisfatti dei filtri immessi si preme su "Search" e inizierà la ricerca effettiva nel sistema. Eventuali risultati verranno mostrati nell'altra tabella sottostante dove sarà possibile aprire una pagina contenente le informazioni dettagliate di un lavoratore facendo doppio clic sulla riga corrispondente oppure pigiare il bottone "Inspect" in alto dopo aver selezionato il profilo desiderato. Per ripulire i parametri di ricerca è possibile utilizzare il tasto "Reset".

Una volta terminato il proprio lavoro il dipendente può disconnettersi dal sistema e tornare alla schermata di login oppure chiudere direttamente l'applicazione.

## 5.2 Software information

In questa parte verrano descritte le varie tecnologie utilizzate per la progettazione, implementazione e testing del software, inoltre verranno specificate delle informazioni generali sul sistema.

### 5.2.1 Software technology

Il sistema software "SeasonalWorkerApp" è stato scritto in *java*, per le interfaccie è stata usata la libreria *javafx* in particolare il linguaggio *fxml*. Per migliorare l'interfaccia sono stati usati dei fogli di stile *css*, per il testing è stata usata la libreria *junit*. Tutte queste tecnologie sono state integrate in un progetto *Maven* dove grazie al lifecycle offerto dal programma è stato possibile compilare, fare testing ed eseguire il progetto in maniera facile. Per la parte di database abbiamo utilizzato *json*, in particolare la libreria *Gson* offre un'interfaccia molto semplice per la gestione di tali file. Per la parte di progettazione è stato utilizzato *UML*, in particolare i diagrammi di sequenza, attività, classi e casi d'uso.

### 5.2.2 System information

Qui di seguito verranno elencate le caratteristiche del sistema:

1. I dipendenti dell'agenzia che possono accedere al sistema verranno inseriti dal responsabile dell'agenzia direttamente nel database, in particolare in ***employers.json***.

2. Se un lavoratore vuole aggiornare i propri dati, deve contattare l'agenzia e fornire il proprio id. Se il lavoratore non possiede un id (smarrimento), il dipendente dell'agenzia potrà identificare il lavoratore tramite i dati anagrafici, quindi tramite la funzionalità *serach*, il dipendente potrà ottenere l'id del lavoratore e successivamente con la funzionalità update potrà aggiornare i dati.