



**Università degli Studi di Verona - Dipartimento di
Informatica**

Corso di Laurea Informatica - Anno Accademico 2020/21

**DOCUMENTAZIONE PROGETTO
INGEGNERIA DEL SOFTWARE**

Sistema software per la gestione delle informazioni socio-sanitarie di monitoraggio della popolazione italiana per la prevenzione di contagi e pandemie.

Mirandola Giacomo - VR429611

Tonin Davide - VR437255

Indice

Indice

[USE CASE & SCHEDE DI SPECIFICA](#)

[SEQUENCE DIAGRAM PER USE CASE](#)

[ACTIVITY DIAGRAM](#)

[CLASS DIAGRAM & SEQUENCE DIAGRAM](#)

[Sviluppo: collaborazione team, progettazione e pattern, scelte progettuali](#)

Collaborazione team di sviluppo

Dashboard fase finale progetto

Documentazione

Progettazione e pattern usati

PATTERN MVC

PATTERN DAO

PATTERN SINGLETON

PATTERN TEMPLATE

Scelte progettuali

Gestione dei dati

Struttura dati Firebase

[Organizzazione della GUI](#)

Login page

Dashboard

Provincia/Regioni/Comuni

Utenti

Contagi Comuni & Decessi Province

Report Decessi

Report Malattie Contagiose

Export

[Test e validazione](#)

Ispezione codice e documentazione

Test degli sviluppatori

Test esterno

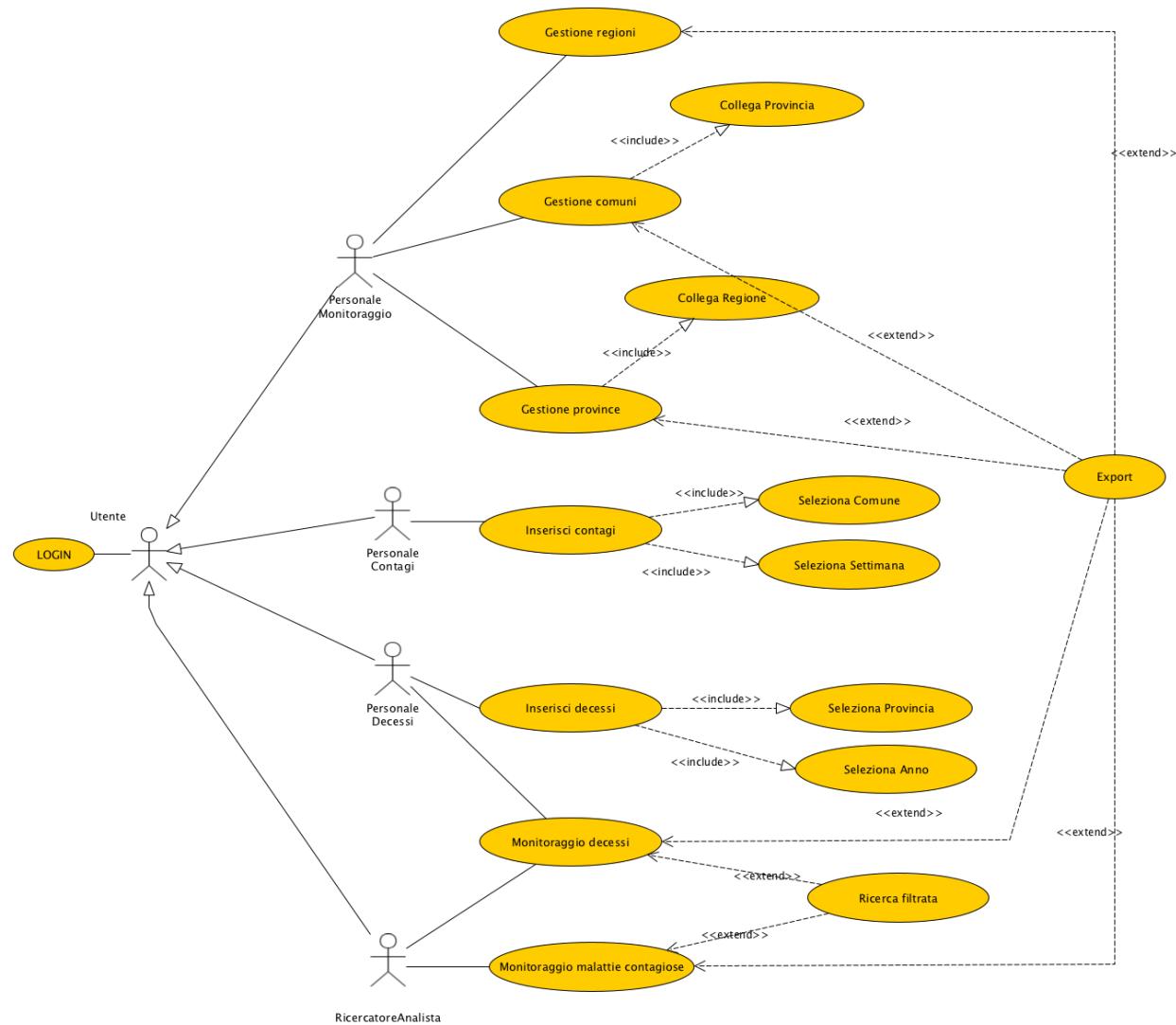
USE CASE & SCHEDE DI SPECIFICA

Solo gli utenti che si autenticano nel sistema possono eseguire delle operazioni, a seconda del ruolo.

Abbiamo identificato 4 ruoli utente nel sistema, più l'amministratore che può fare qualsiasi operazione, e può anche gestire gli utenti.

Ogni utente, quando effettua l'accesso, viene reindirizzato verso una dashboard con le operazioni che può eseguire.

USE CASE



USE CASE ADMIN



SCHEDE DI SPECIFICA

Attori	Personale Monitoraggio
Pre-condizioni	Aver effettuato l'accesso al sistema come personale monitoraggio
Operazioni messe a disposizione	Visualizzazione, inserimento e modifica di comuni, province e regioni
Sequenza	<p>1) L'utente seleziona dal menu una voce fra "Comuni", "Province" e "Regioni" (Prendiamo esempio Regioni, per le altre due è sempre uguale):</p> <p>2) Visualizza tutte le Regioni</p> <p>3) Può effettuare le seguenti operazioni:</p> <ul style="list-style-type: none"> a) Selezionare una regione e modificarla b) Inserire una nuova regione c) Esportare i dati <p>4) Può ripetere in loop le operazioni</p> <p>5) Quando ha finito esegue il logout</p>
Post-condizioni	Nuova successione di operazioni

Attori	Personale Contagi
Pre-condizioni	Aver effettuato l'accesso al sistema come personale contagi
Operazioni messe a disposizione	Inserimento e modifica dei contagi, solo dei comuni di cui è responsabile
Sequenza	<p>1) L'utente seleziona dal menu la voce "Contagi Comuni"</p> <p>2) Seleziona comune e settimana, e carica eventuali dati già inseriti</p> <p>3) Inserisce/Modifica i contagi per le varie malattie contagiose</p> <ul style="list-style-type: none"> a) Per alcune malattie contagiose, può inserire delle complicazioni <p>4) Può ripetere in loop le operazioni</p> <p>5) Quando ha finito esegue il logout</p>
Post-condizioni	Nuova successione di operazioni

Attori	Personale Decessi
Pre-condizioni	Aver effettuato l'accesso al sistema come personale decessi
Operazionimesse	Inserimento e modifica dei decessi per provincia, visualizzare i report per i decessi
Sequenza	<p>1) L'utente seleziona dal menu la voce "Decessi Province"</p> <p>a) Seleziona provincia e settimana, e carica eventuali dati già inseriti</p> <p>b) Inserisce/Modifica i decessi per le varie cause di morte</p> <p>b) Può ripetere in loop le operazioni</p> <p>d) Quando ha finito esegue il logout</p> <p>2) L'utente seleziona dal menu uno dei report decessi (per provincia, regione, nazione):</p> <p>a) Inserisce i filtri richiesti</p> <p>b) Visualizza i report</p> <p>c) Può esportare i dati</p> <p>3) Quando ha finito esegue il logout</p>
Post-condizioni	Nuova successione di operazioni

Attori	Ricercatore Analista
Pre-condizioni	Aver effettuato l'accesso al sistema come ricercatore analista
Operazioni permesse	Monitoraggio malattie contagiose e decessi
Sequenza	<p>L'utente può vedere i vari report indifferentemente dall'ordine.</p> <p>1) Seleziona dal menu uno dei report malattie contagiose (per provincia, regione, nazione): dopo aver inserito i filtri richiesti, può visualizzare i dati aggregati confrontando contagi e decessi per ogni malattia contagiosa. Può effettuare l'export dei dati selezionati in vari formati.</p> <p>2) Seleziona dal menu il report contagi comuni: dopo aver inserito i filtri richiesti, può visualizzare i dati aggregati per i comuni scelti.</p> <p>3) Seleziona dal menu uno dei report decessi (per provincia, regione, nazione): dopo aver inserito i filtri richiesti, può visualizzare i dati aggregati per le varie cause di morte. Può effettuare l'export dei dati selezionati in vari formati.</p> <p>4) Quando ha finito esegue il logout</p>
Post-condizioni	Nuova successione di operazioni

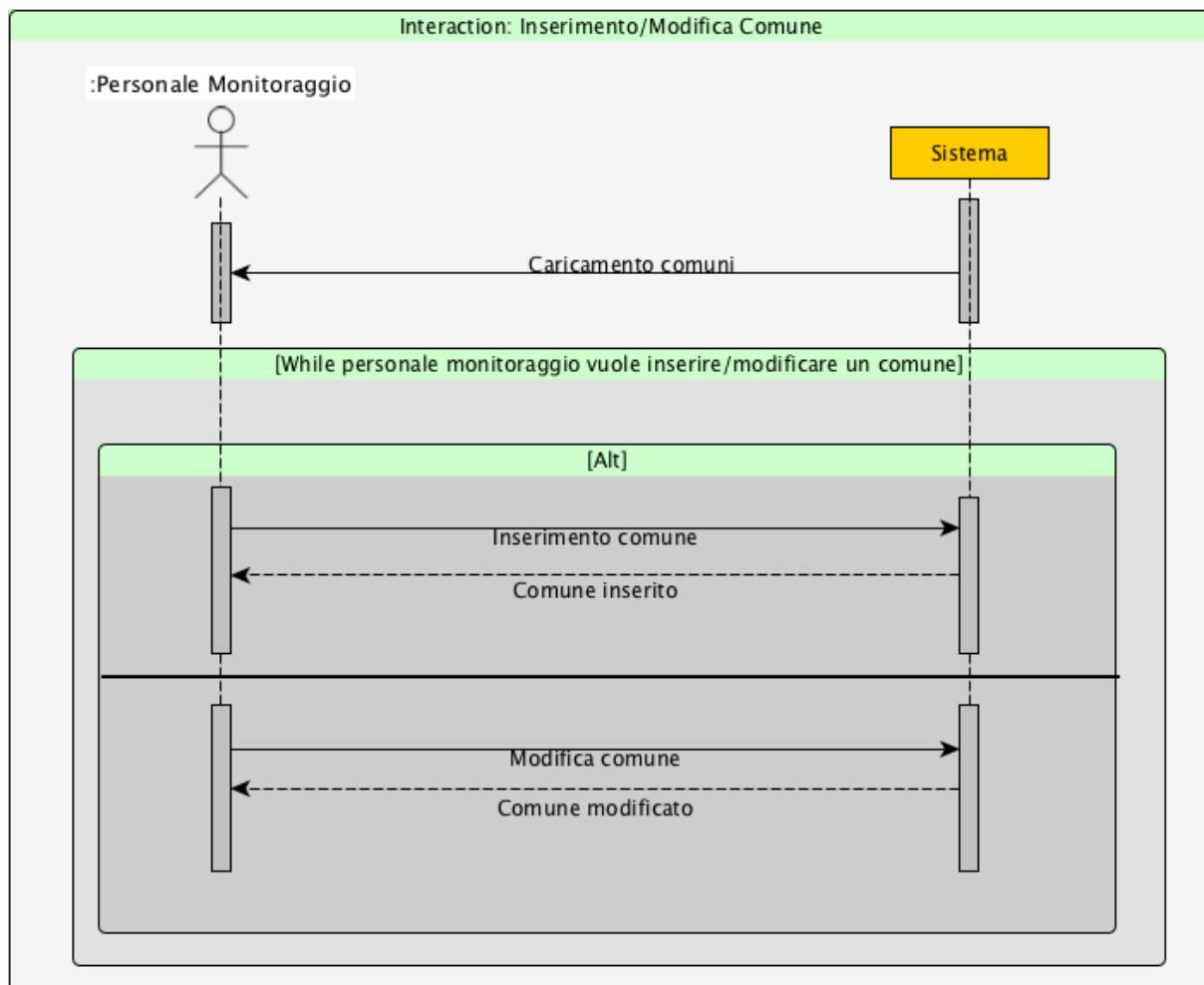
L'utente **admin** ha i permessi e la visibilità su tutte le interfacce, quindi può fare tutto quanto scritto sopra, oltre a inserire o modificare gli utenti.

SEQUENCE DIAGRAM PER USE CASE

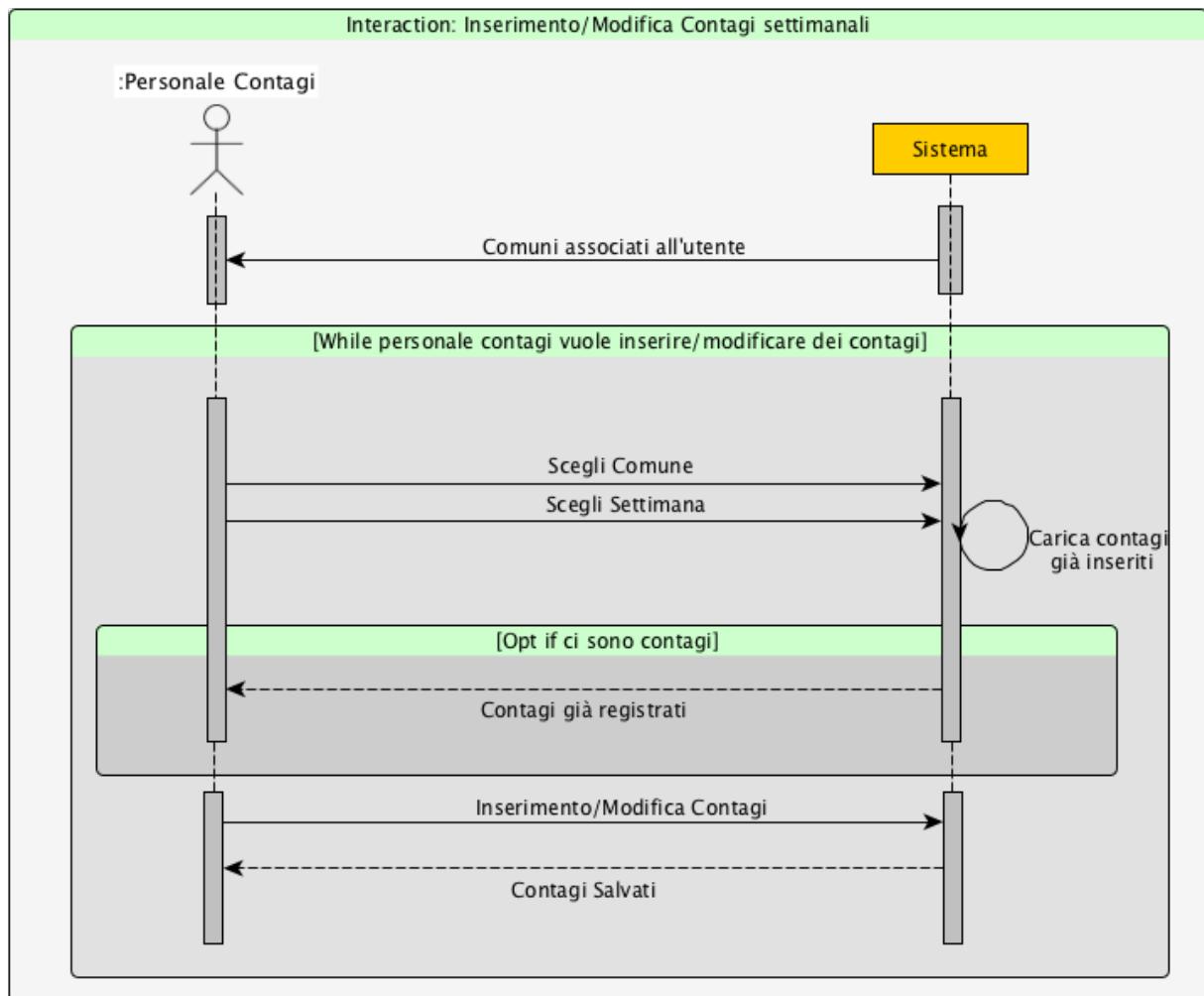
Abbiamo creato i sequence diagram per i principali use case, omettendo alcuni casi in cui le operazioni sono molto simili.

Sequence diagram personale monitoraggio

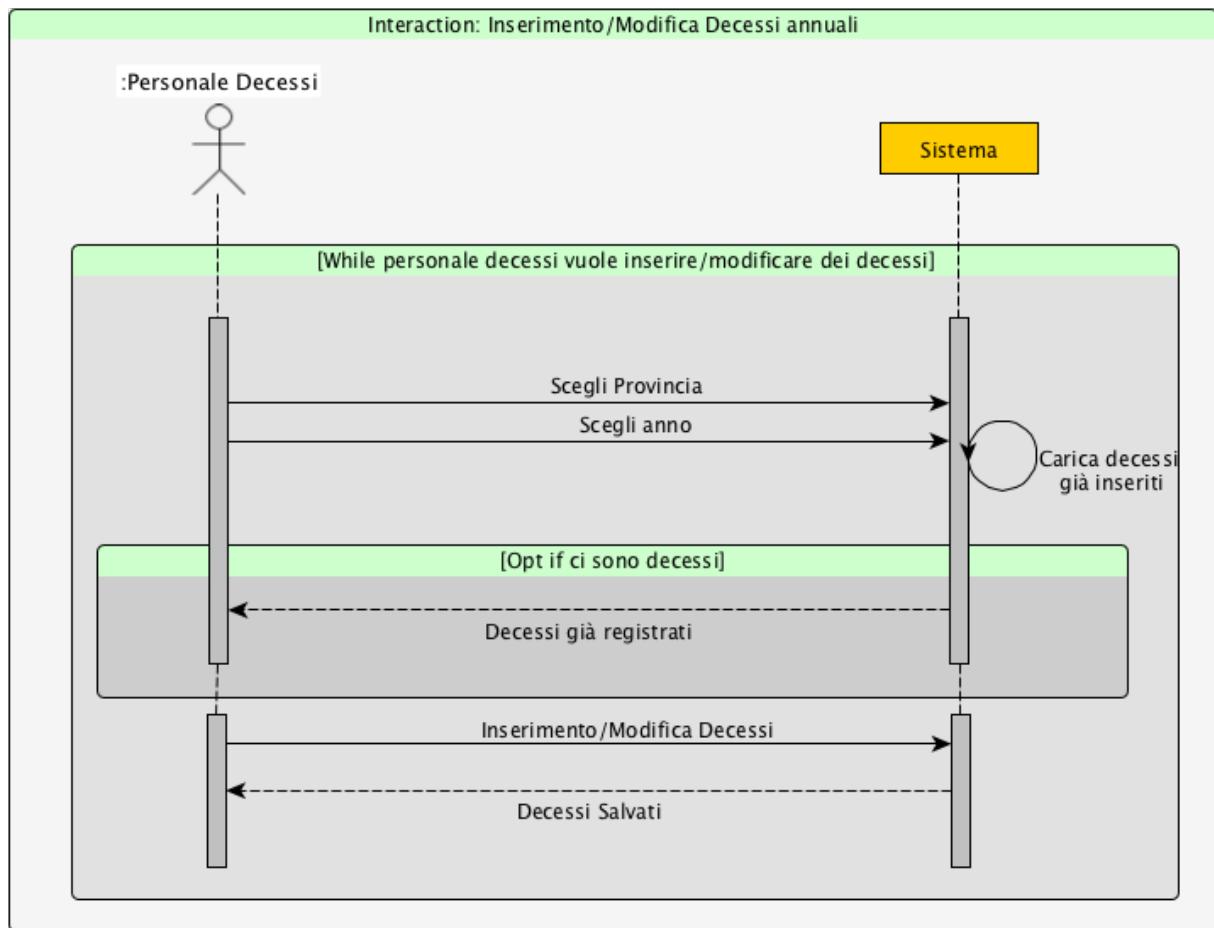
Lo stessa sequenza è prevista per l'inserimento e la modifica di regione e provincia



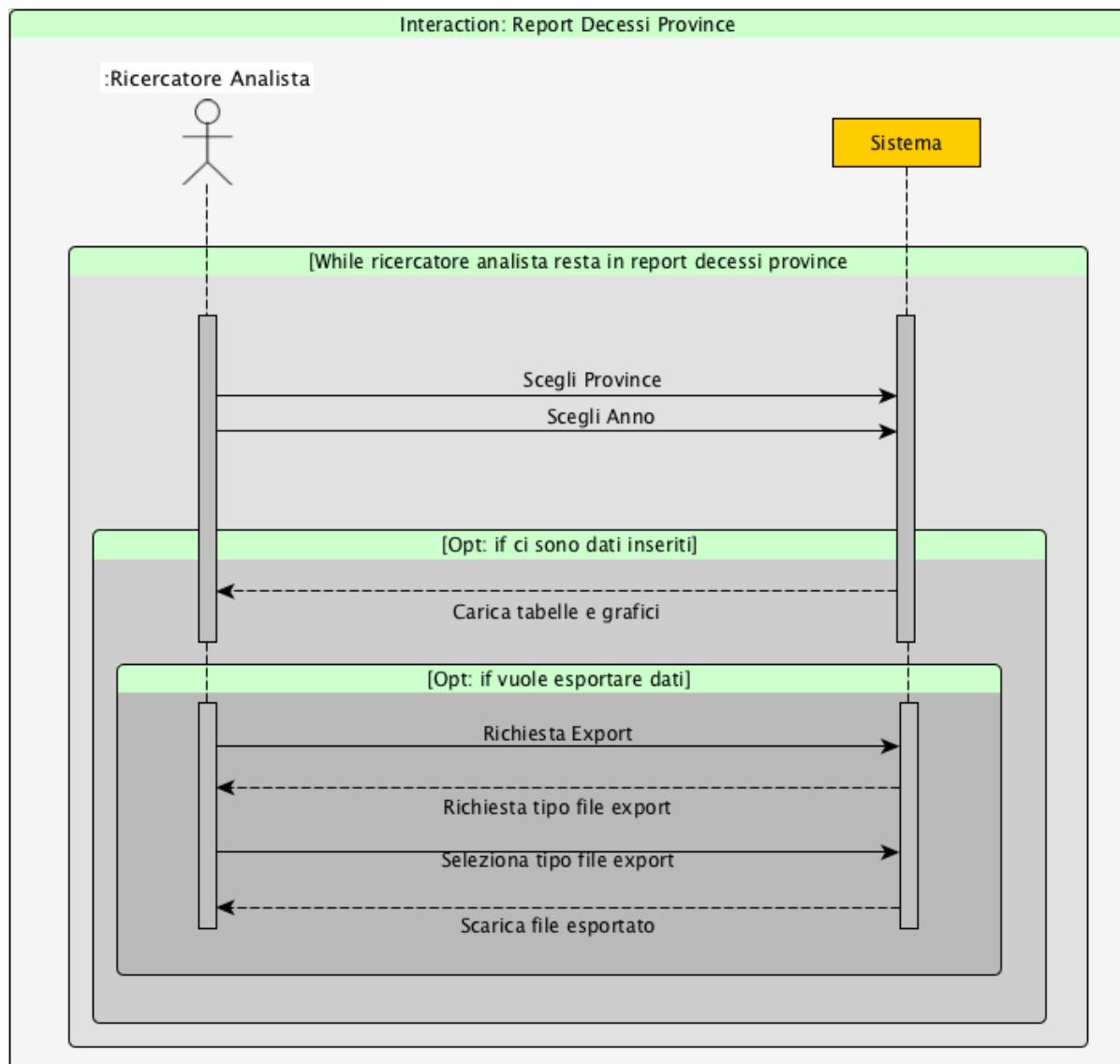
Sequence diagram personale contagi - inserimento/modifica contagi settimanali

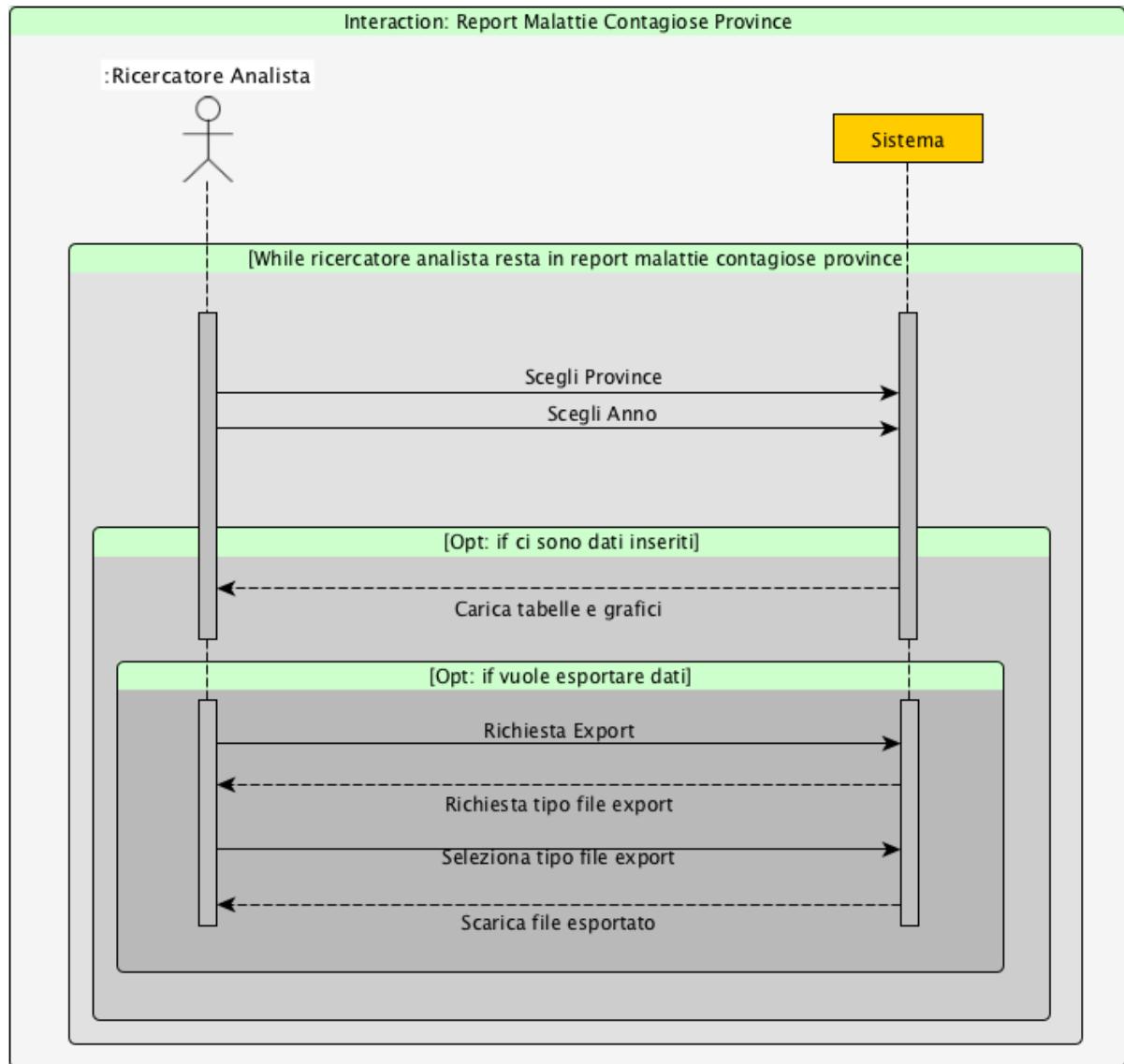


Sequence diagram personale decessi - inserimento/modifica decessi annuali



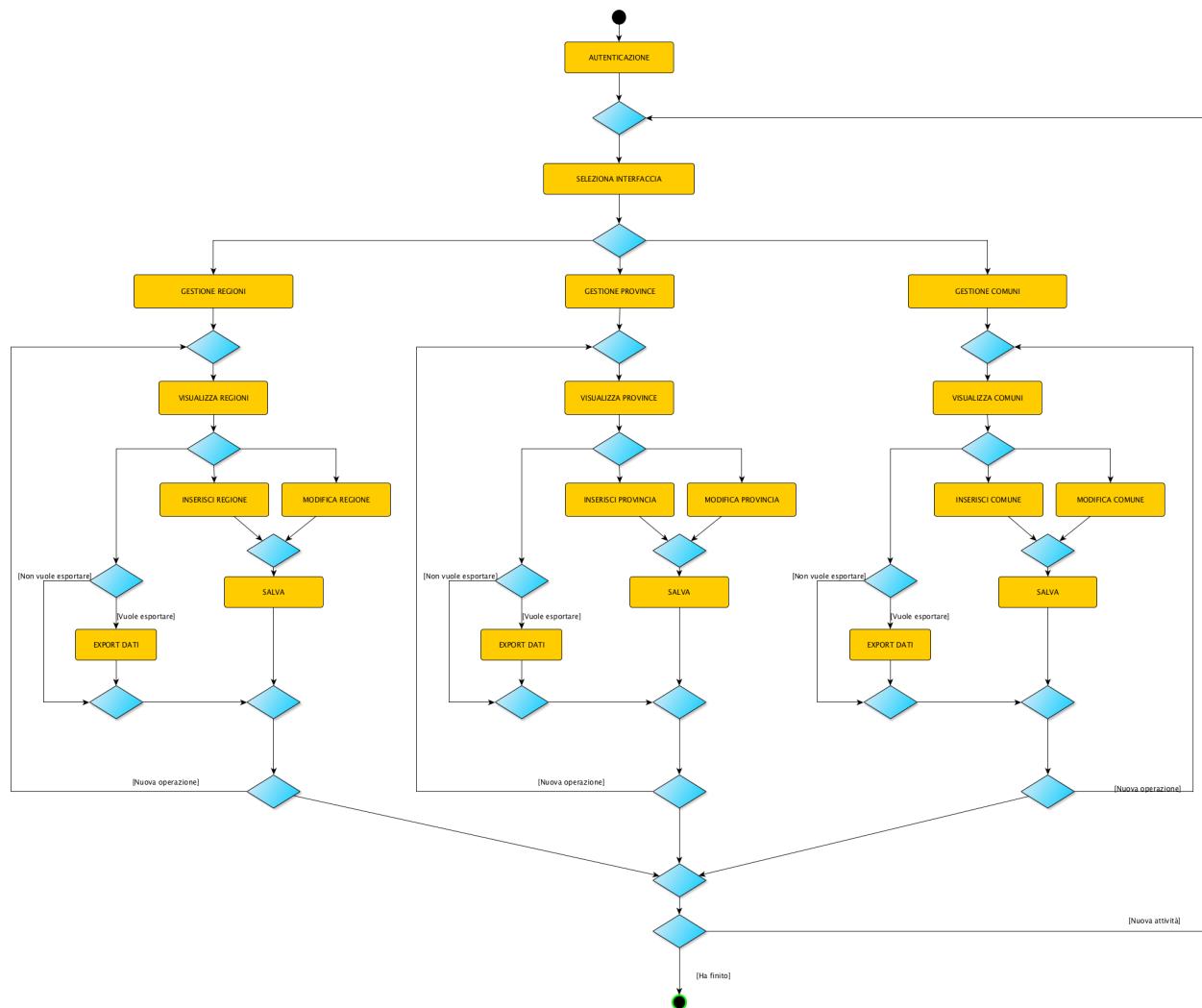
Sequence diagram ricercatore analista - report decessi province e malattie contagiose province



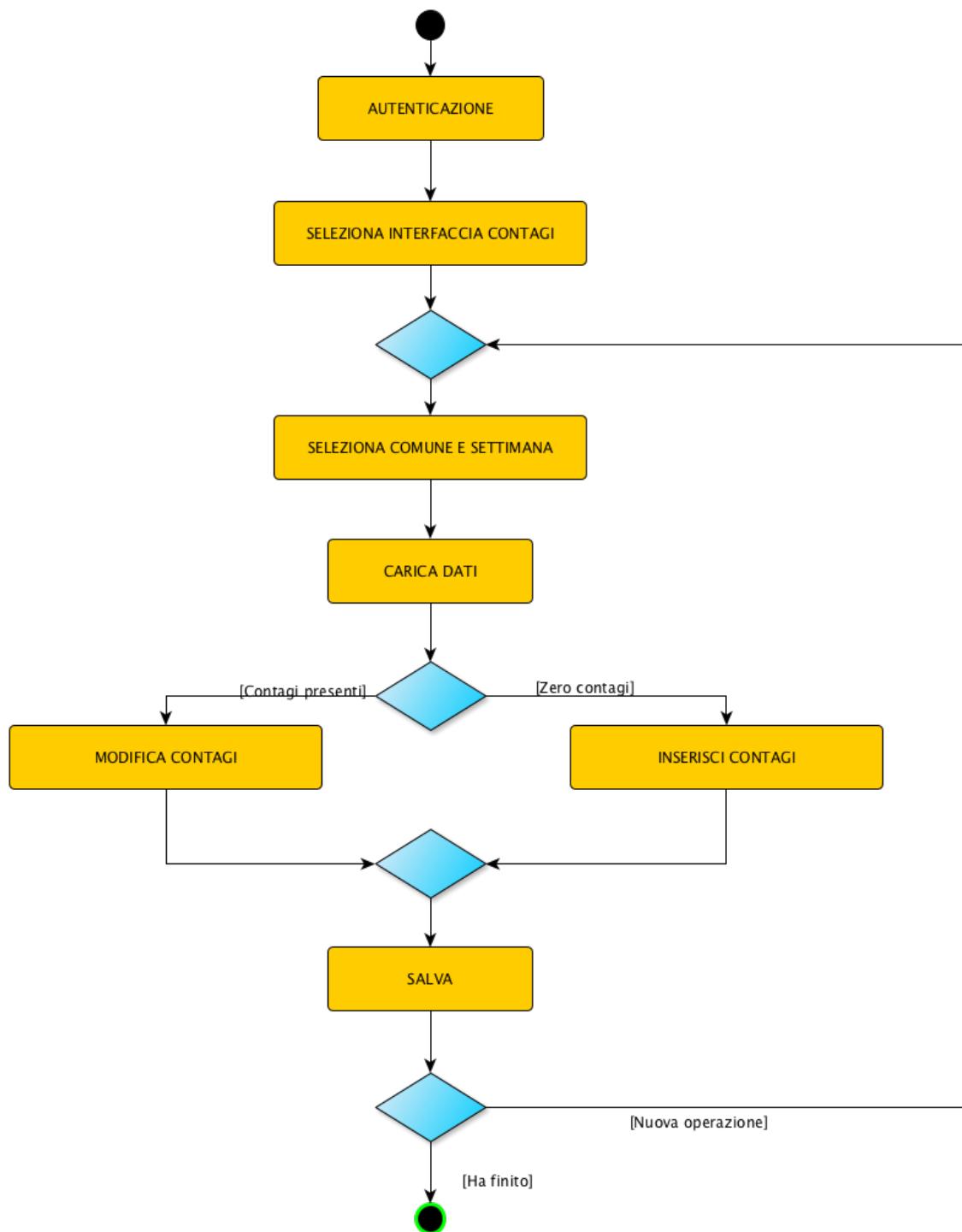


ACTIVITY DIAGRAM

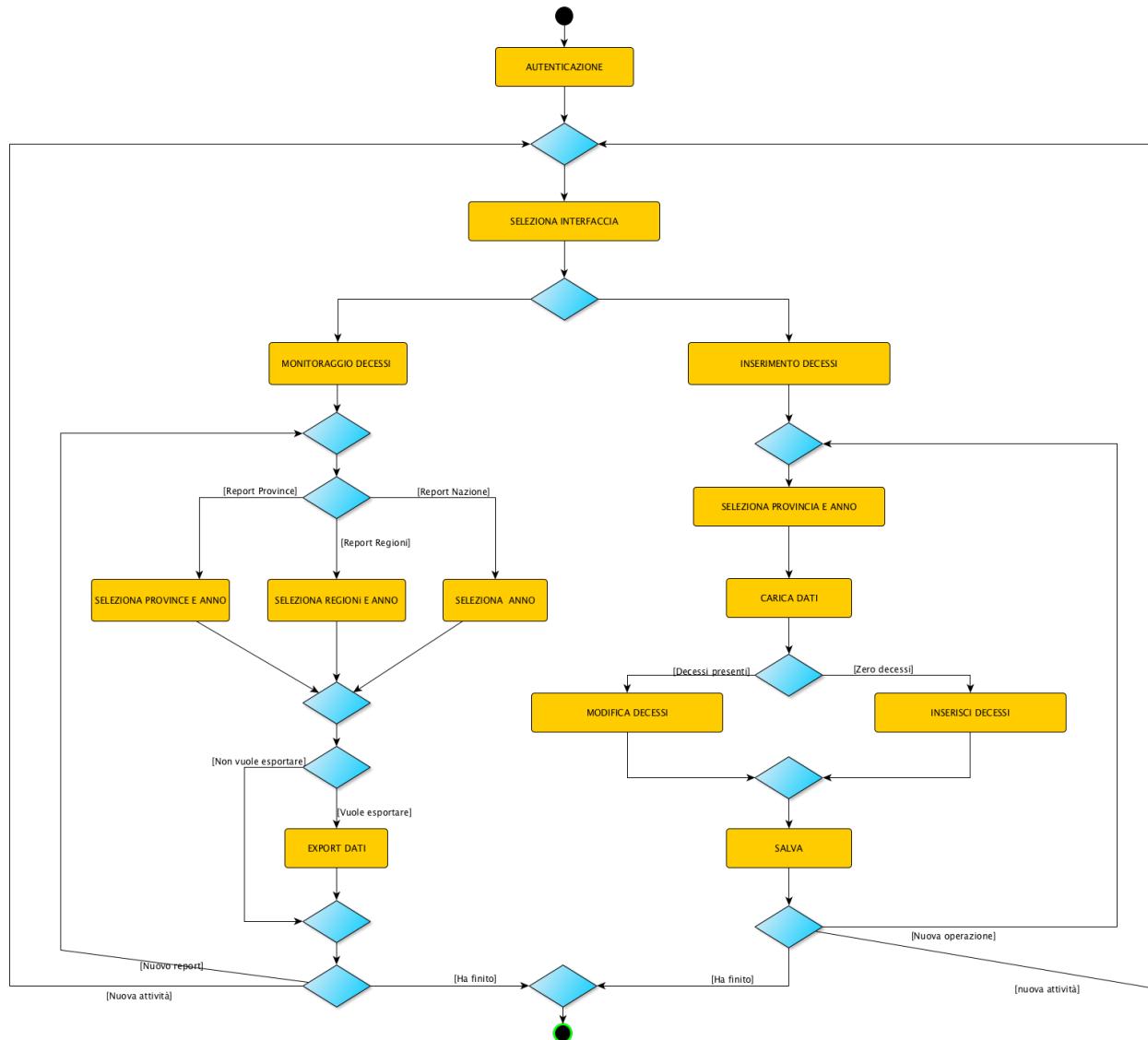
Activity Diagram personale monitoraggio



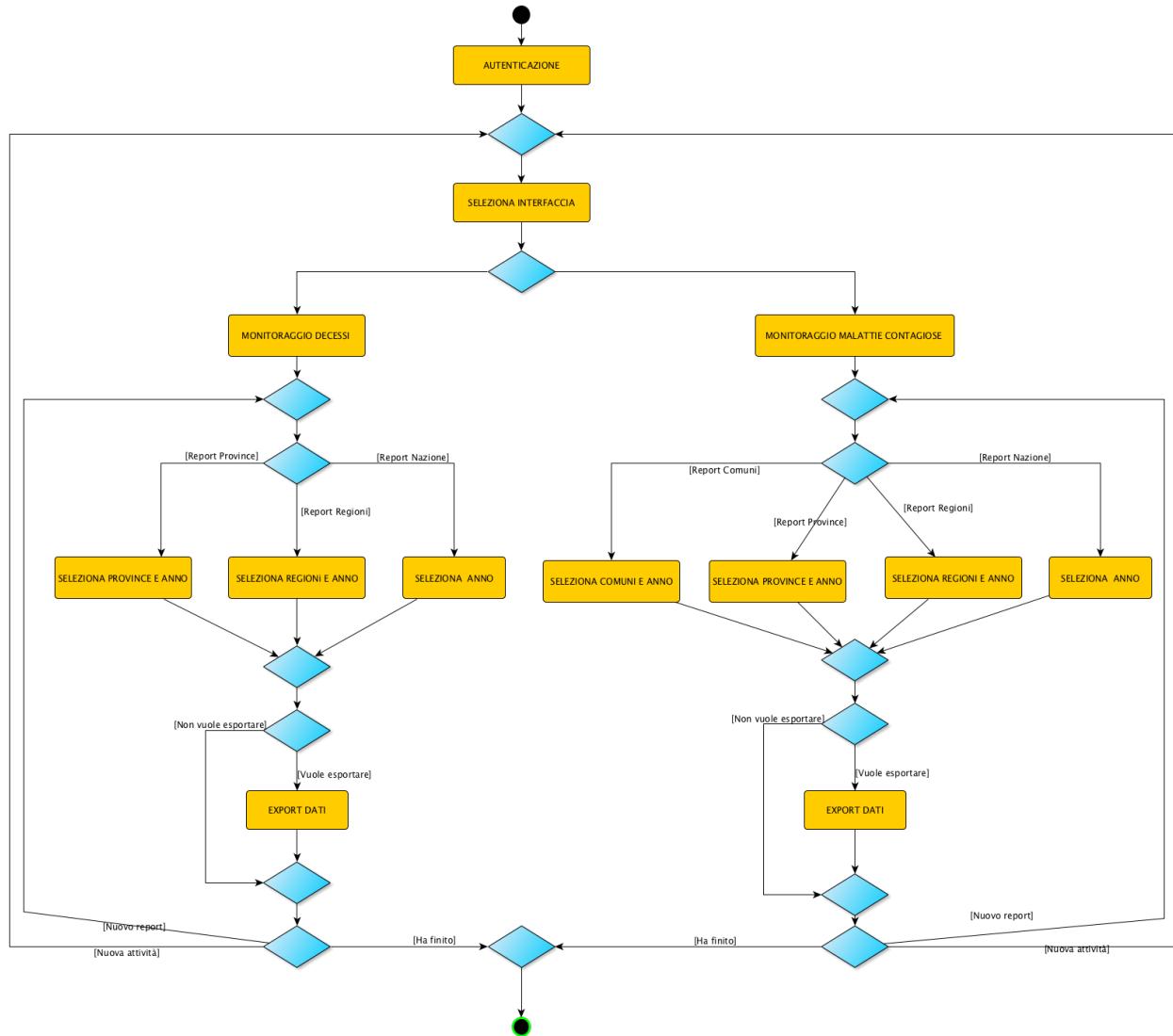
Activity Diagram personale contagi



Activity Diagram personale decessi

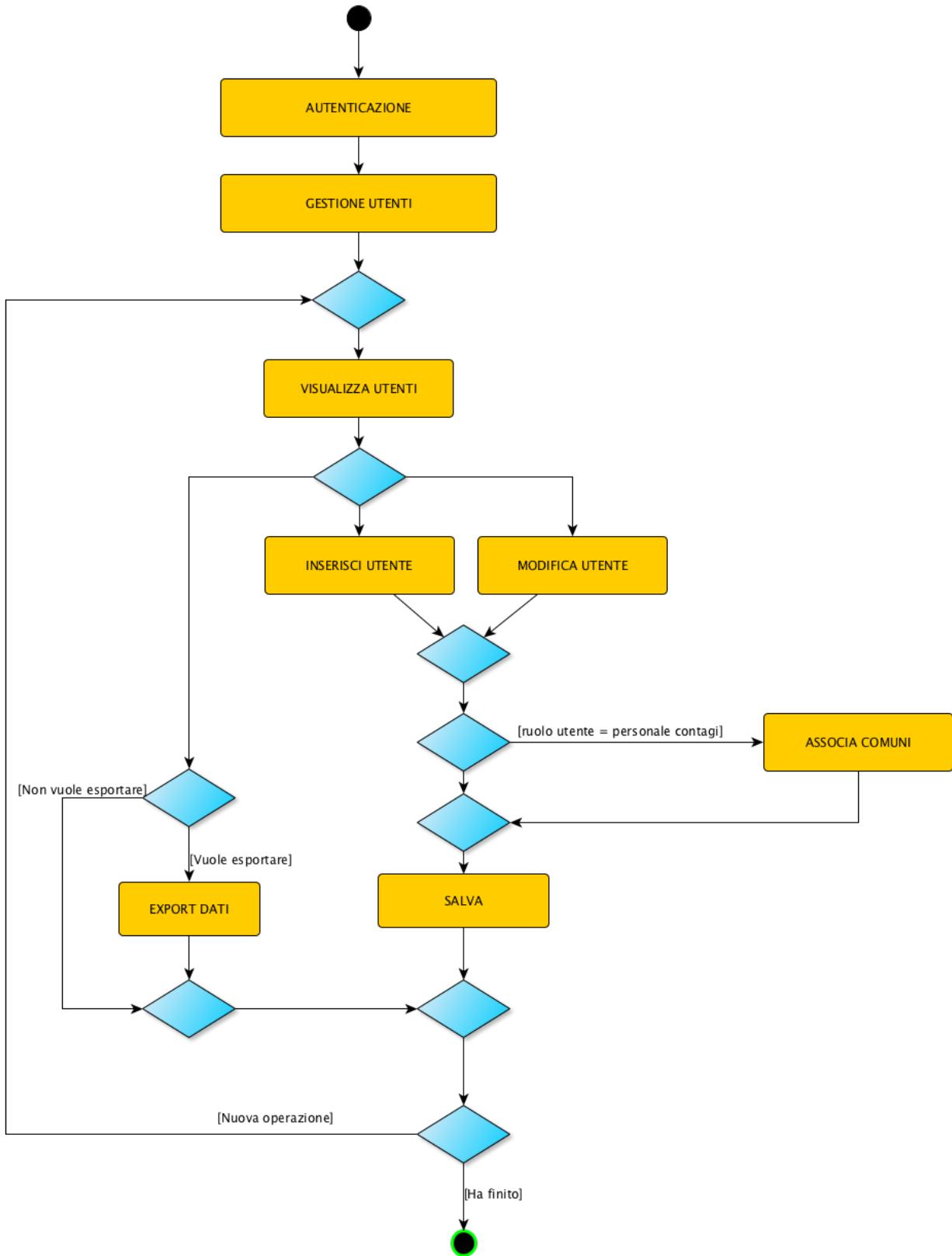


Activity Diagram ricercatore analista



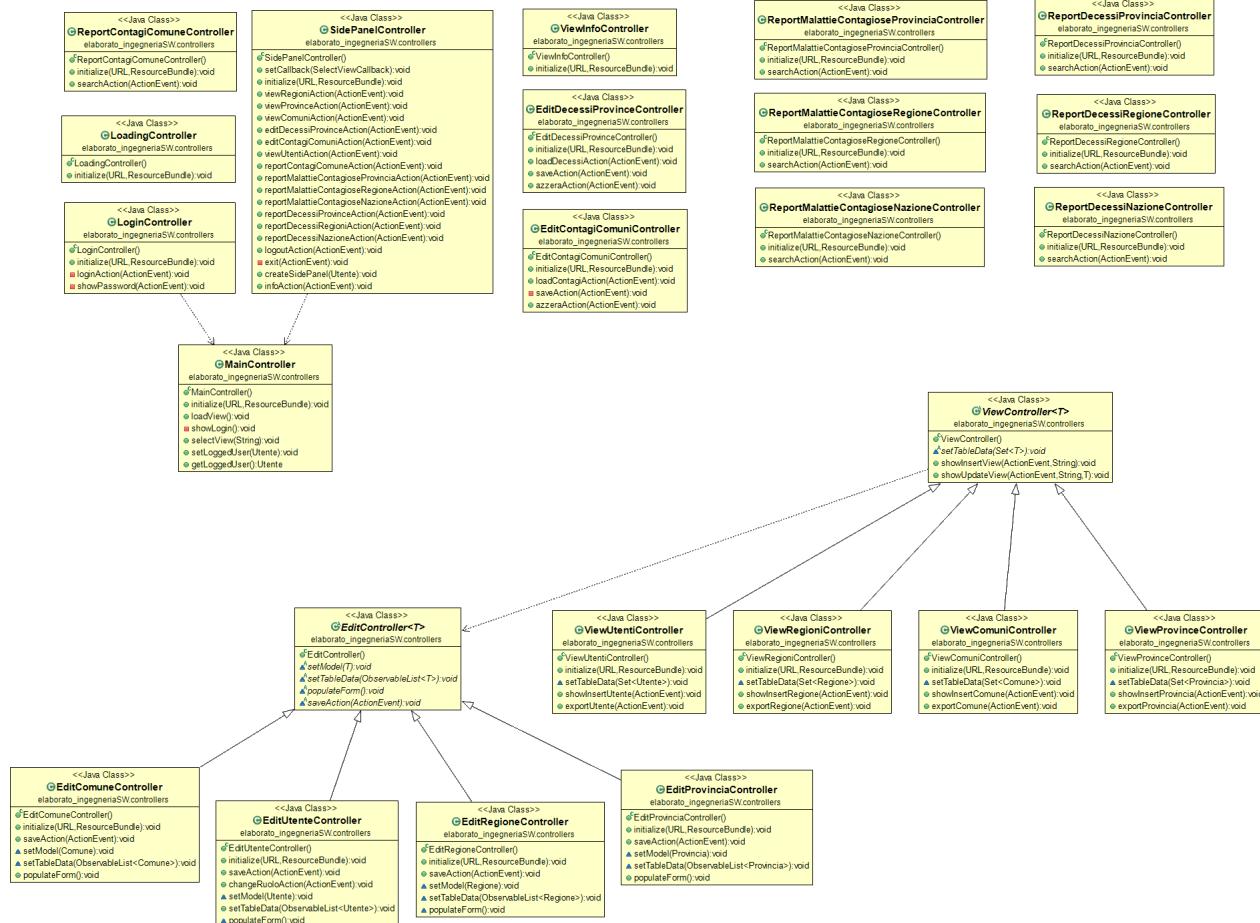
Activity Diagram utente admin

L'utente admin ha tutti i permessi, abbiamo descritto solamente l'attività di gestione utenti per non ripetere le operazioni già presenti negli altri diagrammi.

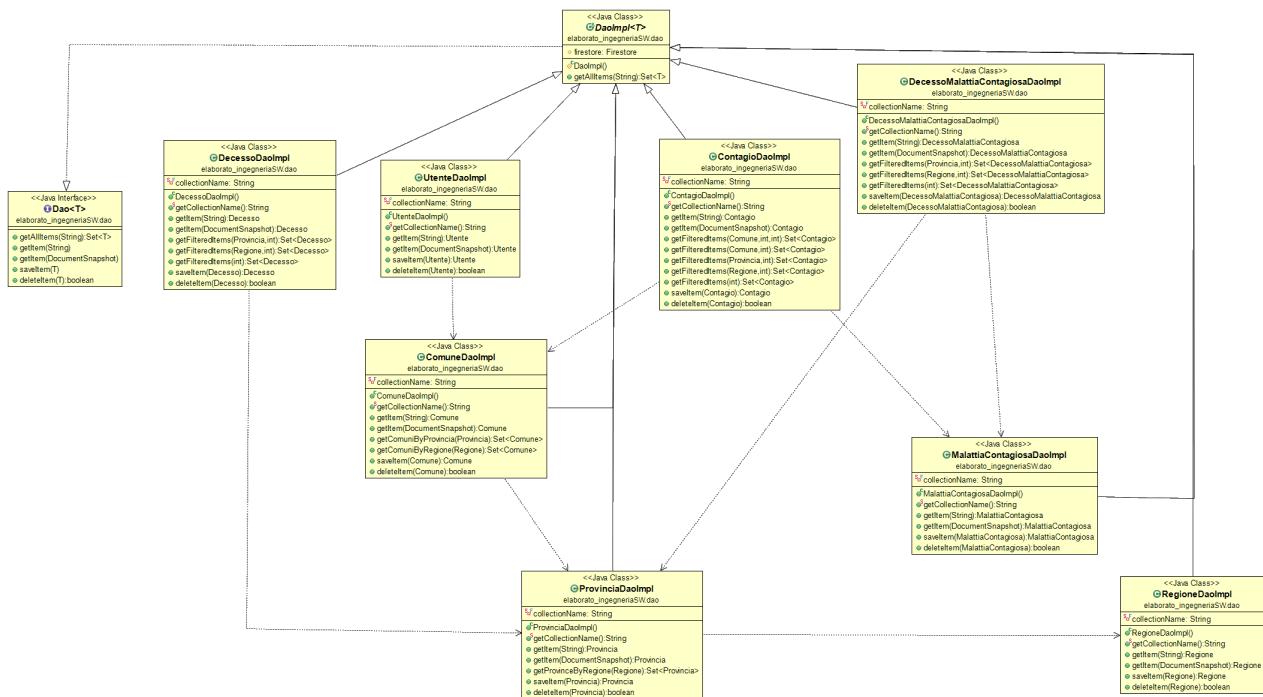


CLASS DIAGRAM & SEQUENCE DIAGRAM

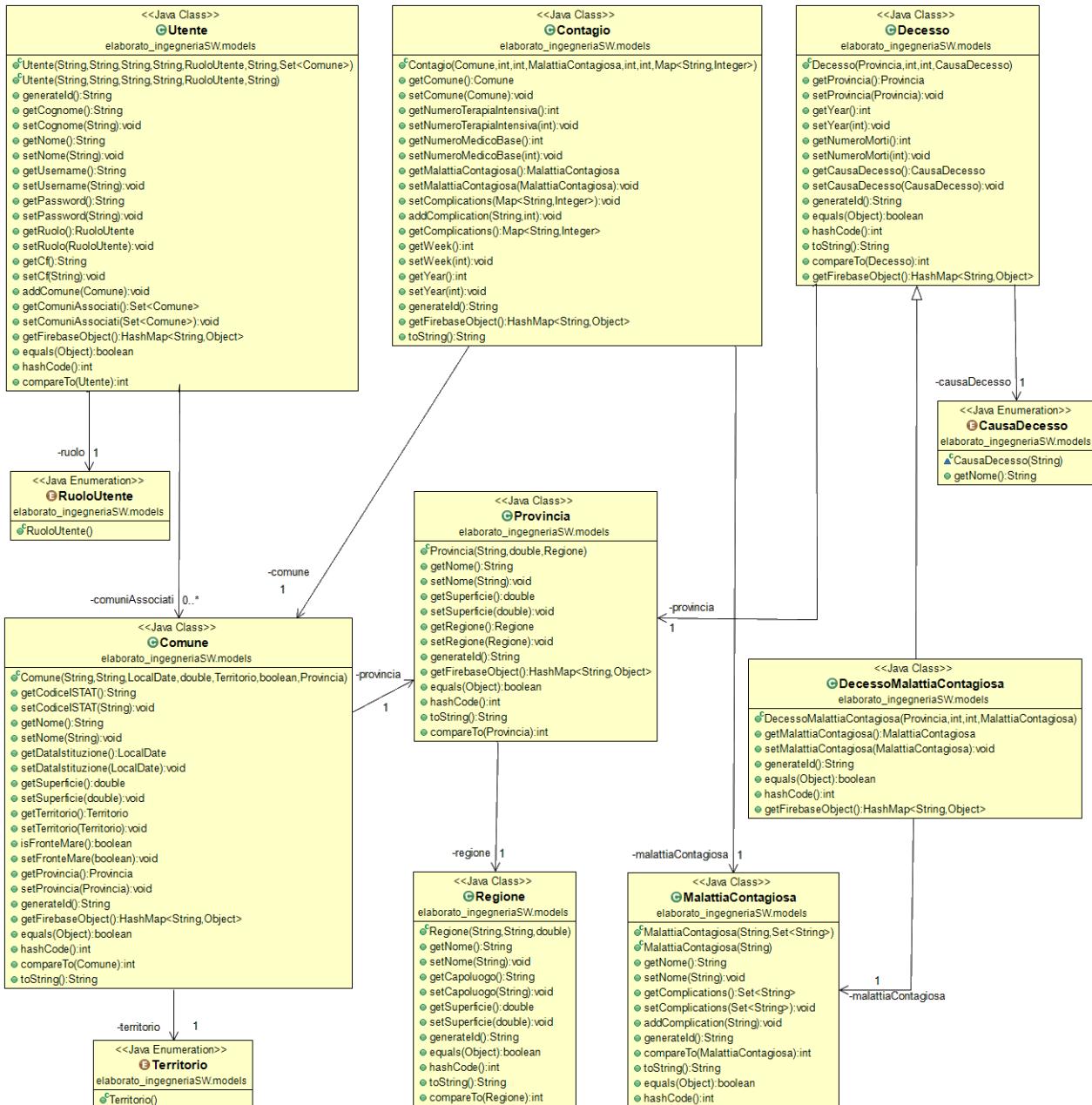
Class Diagram dei Controller



Class Diagram dell'implementazione DAO

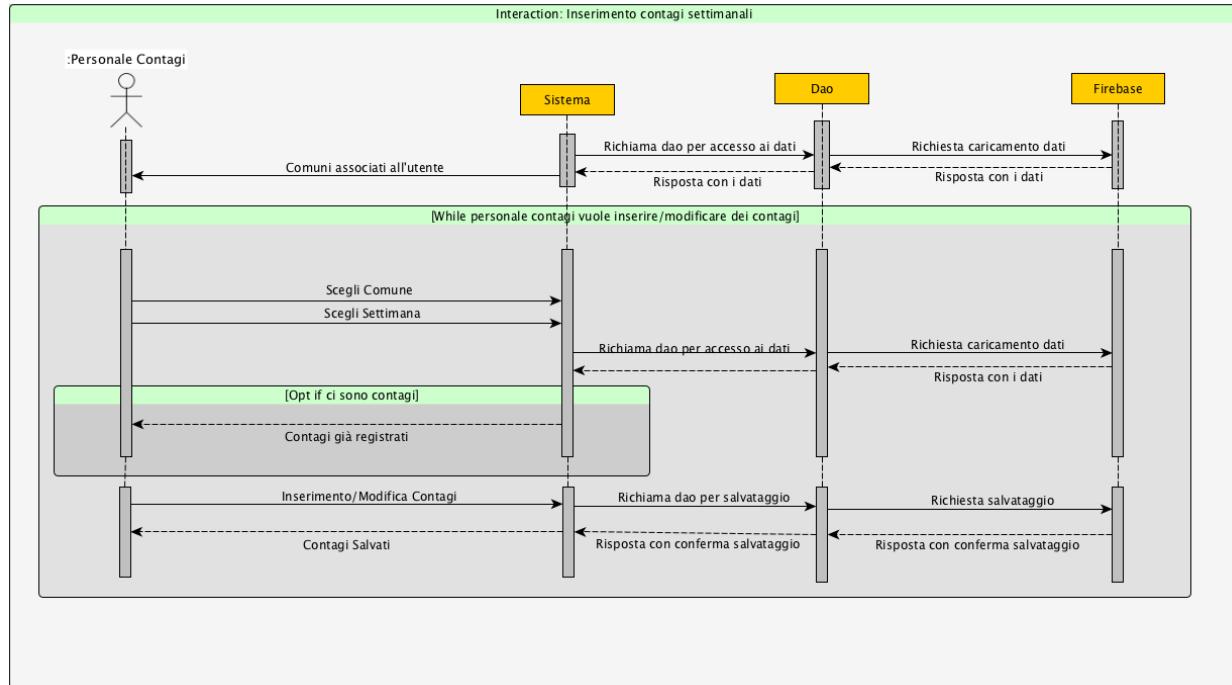


Class Diagram dei Models dell'applicazione



Sequence Diagram

Abbiamo scelto il seguente sequence diagram perchè racchiude varie funzionalità del sistema, in particolare l'implementazione di un DAO, l'interazione con il database su Firebase e l'interazione utente sistema tramite interfaccia grafica



Sviluppo: collaborazione team, progettazione e pattern, scelte progettuali

Introduzione

Il progetto realizzato è un'applicazione java desktop sviluppata principalmente con la libreria **JavaFX**, e l'utilizzo di due librerie di supporto per la grafica e componenti aggiuntivi **JFoenix** e **ControlsFX**.

Per sviluppare tale software è stato utilizzato l'ambiente di sviluppo [IntelliJ Idea](#), insieme a [SceneBuilder](#) per la progettazione dell'interfaccia grafica.

Il processo di sviluppo è stato coordinato secondo il metodo **AGILE** e **SCRUM**. Il team si trovava settimanalmente per meeting solitamente di media/breve durante, nei quali lo scopo spaziava dal progettare concettualmente la base di dati e l'app fino ad individuare le criticità del sistema testandolo al momento. Alcuni incontri duravano più a lungo, perchè in alcuni casi nascevano delle discussioni per cercare di trovare la migliore soluzione a fronte di idee diverse su un certo

argomento.

Durante i *daily scrum* l'obiettivo era anche pianificare non dei "daily sprint" ma dei "weekly sprint", nel nostro caso, con modifiche, migliorie e le nuove implementazioni del sistema. Il gruppo, vista la situazione attuale legata al COVID-19, ha optato per *scrum-meeting* sulle piattaforme **Zoom** ed **AnyDesk**, oltre ad alcuni incontri telefonici o nell'ufficio di lavoro.

La teoria alla base del metodo utilizzato è quella del controllo empirico dei processi, secondo la quale, da un lato, la conoscenza deriva dall'esperienza e, dall'altro lato, le decisioni si basano su ciò che si conosce. Per questo motivo si prevede un processo iterativo con un approccio incrementale che ottimizza, passo dopo passo (e sprint dopo sprint), la prevedibilità ed il controllo del rischio.

Collaborazione team di sviluppo

Git & Github



Per la condivisione del codice ed il controllo di versione, abbiamo scelto **git**, in particolare abbiamo creato una repository ospitata su **Github**. LINK REPOSITORY: https://github.com/Mirgiacomo/elaborato_ingegneriaSW

Il codice nel repository è stato organizzato in branch diversi per ogni attività, creati in base ai vari meeting e necessità che nascevano durante lo sviluppo.

Inoltre, sono stati creati due branch principali:

- `test` : branch di partenza e fine delle varie attività
- `master` : branch utilizzato per spostare le parti testate e funzionanti dal branch di test

In questo modo, ogni componente del team si occupava di una certa attività, eventualmente chiedendo un aiuto o discutendo su vari dubbi, e una volta terminata il codice relativo veniva spostato nel branch di test affinché fosse disponibile per tutti e per effettuare i test con altre parti aggiunte.

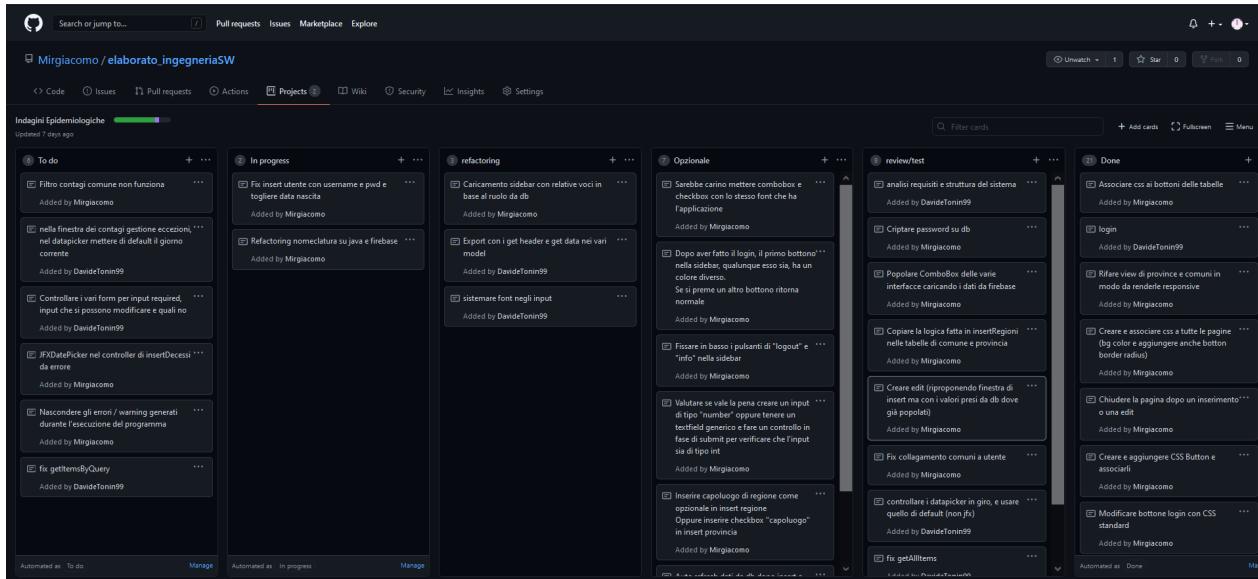
GitHub project

Abbiamo scelto di utilizzare un **tool di project management** web-based, che ci ha aiutato ad organizzare e gestire le varie richieste, attività (ovvero le user-story), bug e idee.

Fra i vari strumenti disponibili, abbiamo scelto di utilizzare **GitHub Project**, in quanto connesso direttamente con il repository del progetto.

GitHub Project è un tool integrato nella piattaforma di GitHub, gratuito e basato sulle **kanban-board**, con lo scopo appunto di facilitare l'organizzazione del lavoro, la gestione dei task nel tempo e la comunicazione fra gli sviluppatori.

Dashboard fase finale progetto



Ciclo di vita di un task:

1. **To Do**: il punto di partenza, dove vengono segnati i task da sviluppare nell'immediato futuro
2. **In Progress**: segnala che un task è attualmente in fase di sviluppo
3. **Refactoring**: degli sviluppi/task che sono stati implementati e che hanno bisogno di un refactoring dopo un primo sviluppo.
4. **Opzionale**: spazio usato per abbozzare idee che vengono agli sviluppatori e/o in fase di briefing. Queste idee vengono implementate gradualmente nella fase finale del progetto, quando si ha un **prototipo completo e funzionante**
5. **Review/Test**: quando uno sviluppatore finisce un task lo segnala agli altri sviluppatori che devono fare la "review" del nuovo codice. Se il nuovo codice sviluppato passa la review, allora viene messo in Done, se invece non passa la review si cerca di capire per quali motivi non è andato a buon fine, in modo da identificare eventuali bug. Questo step è necessario per evitare di **confermare** codice con errori o incoerenze
6. **Done**: il punto di arrivo del task, segnala che il task è stato completato e soprattutto **testato** con successo

Zoom & AnyDesk & Telefono

I weekly scrum sono stati svolti sulla piattaforma online Zoom o per telefono con incontri settimanali, più o meno frequenti a seconda degli impegni e soprattutto delle **idee o problemi riscontrati**, per confrontarci, fare il punto della situazione e decidere a cosa dare priorità nella parte di sviluppo. Inoltre, un altro strumento utilizzato è stato AnyDesk, per fare una sorta di *pair-programming* che risulta essere molto interessante ed importante, in quanto da la possibilità di aiutarsi a vicenda nella scrittura o spiegazione del codice.

Documentazione





La documentazione del progetto è stata scritta in formato Markdown con l'aiuto dei software [stackedit.io](#), un tool esterno per editare Markdown con più facilità, e [Typora][<https://typora.io/>].

Markdown perchè è utilizzato spesso per la scrittura di testi formattati e in modo semplice, soprattutto nella documentazione e in particolare il file README dei repository su GitHub. Inoltre, il file può essere poi convertito in altri formati molto utili, tra cui `html` (se per esempio lo si vuole formattare diversamente) e `pdf`.

yED & Eclipse

Per lo sviluppo dei vari diagrammi abbiamo utilizzato il software yED ed Eclipse per la generazione del diagramma delle classi

Progettazione e pattern usati

PATTERN MVC

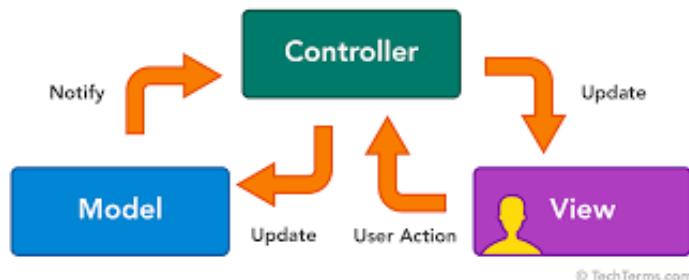
MVC - Model View Controller

La scelta del pattern MVC è stata fortemente condizionata dall'utilizzo del tool SceneBuilder, dato che esso produce una view, ovvero un file FXML il quale viene collegato ad un controller, ovvero una classe java. Questo mostra come il tool SceneBuilder sia fortemente orientato verso il pattern MVC. Il team di sviluppo, una volta presa familiarità con SceneBuilder, ha deciso di continuare con esso e quindi di implementare il pattern MVC.

DESCRIZIONE

In breve, il pattern architetturale MVC divide il sistema in 3 componenti che interagiscono tra loro:

- **Model:** definisce come viene rappresentata, gestita e memorizzata l'informazione
- **View:** definisce la rappresentazione a video dei dati e delle interfacce
- **Controller:** definisce la logica applicativa, ovvero il comportamento del sistema rispetto ad input esterni, per esempio dell'utente



Abbiamo tenuto suddivisi fin da subito i file di model, view e controller inserendoli in tre cartelle apposite del progetto.

Ogni view è collegata ad un controller, che tramite il DAO accede ai dati salvati su Firebase, trattandoli come models specifici.

PATTERN DAO

DAO - Data Access Object

Il pattern DAO, ovvero Data Access Object, viene usato per separare le operazioni di accesso ai dati di basso livello da servizi di alto livello. Quindi rende disponibili un set di metodi di "alto livello" accessibili dall'utente, che al loro interno interagiscono a basso livello con i dati. Tale implementazione interna dei metodi non è di competenza dell'utente, che si limita a richiamarli.

Nella sezione di class diagram, c'è lo schema di implementazione del pattern dao.

Ogni implementazione specifica estende una classe astratta generale, in cui è stato definito un metodo parametrizzato per ottenere tutti i dati di una certa collezione, mentre per le operazioni specifiche, ogni Dao implementa i vari metodi richiesti.

INTERFACCIA DAO

```
public interface Dao<T> {
    Set<T> getAllItems(String collectionName) throws ExecutionException,
    InterruptedException;
    T getItem(String itemId) throws ExecutionException, InterruptedException;
    T getItem(DocumentSnapshot document) throws ExecutionException,
    InterruptedException;
    T saveItem(T item) throws ExecutionException, InterruptedException;
    boolean deleteItem(T item);
}
```

PATTERN SINGLETON

SINGLETON

Viene usato per fare in modo che ci sia al più un'istanza di una particolare classe. Le classi singleton vengono sviluppate con un costruttore privato e un metodo pubblico e statico che ritorna un riferimento all'unica istanza della classe.

Il pattern singleton usato nella connessione a Firebase è di fondamentale importanza perché assicura una sola connessione al database durante l'esecuzione dell'app, senza dover aprirla/chiuderla ogni qualvolta si deve accedere al database.

PATTERN TEMPLATE

TEMPLATE

Nel pattern template, una classe astratta espone un template per l'esecuzione dei suoi metodi.

Le sotto-classi che estendono questa classe astratta implementano i metodi a seconda del caso specifico, ma seguendo sempre una struttura ben definita, oppure utilizzando metodi definiti di default dalla classe astratta.

ESEMPIO

```

public abstract class EditController<T> {
    protected ObservableList<T> tableData;
    protected T model;

    abstract void setModel(T model);
    abstract void setTableData(ObservableList<T> tableData);
    abstract void populateForm();
    abstract void saveAction(ActionEvent event);
}

```

Abbiamo utilizzato il pattern template per implementare delle viste molto simili, le viste di view e di edit (utenti, comuni, province, regioni), in modo da avere uno schema da seguire e tenere il codice il più ordinato possibile.

Scelte progettuali

Perchè JAVA?

Avendo già dell'esperienza nell'ambito web, per lo sviluppo di questo progetto abbiamo scelto di utilizzare JAVA e JAVAFX perchè entrambi volevamo imparare la progettazione di interfacce grafiche per applicazioni con queste tecnologie in un contesto diverso, ovvero delle applicazioni Desktop.

Gestione dei dati

FIREBASE



Tra le varie alternative per la gestione e il salvataggio dei dati, tra cui la serializzazione su file o l'implementazione di un database, dopo varie discussioni e considerazioni, avendo già utilizzato in precedenza la serializzazione su file e database relazionali, è stato deciso di utilizzare una tecnologia nuova per noi, ovvero **database NoSql** di Google, **FIREBASE**.

Questo database è conosciuto soprattutto per l'implementazione di applicazioni realtime ed è ospitato nel Cloud Google, che consente di archiviare e sincronizzare i dati fra i vari utenti in tempo reale, oltre a fornire altri aspetti importanti tra cui flessibilità e scalabilità.

L'aspetto della sincronizzazione in tempo reale è molto importante quando si parla di monitoraggio in ogni ambito, dal monitoraggio di un impianto industriale al monitoraggio della popolazione per la prevenzione di epidemie, perchè consente di avere un'istantanea della situazione ed effettuare analisi e previsione in base ai dati raccolti.

Inoltre, Firebase offre due alternative: una soluzione basata su json (Realtime database), e una su documenti organizzati in raccolte (Firestore Cloud).

Noi abbiamo scelto di utilizzare la seconda, **Firebase Cloud**, perchè molto comoda per l'organizzazione dei dati e la connessione con **models** (vedi pattern MVC) utilizzati nell'applicazione Java.

Link alla dashboard per il [progetto di Firebase](#)

Struttura dati Firebase

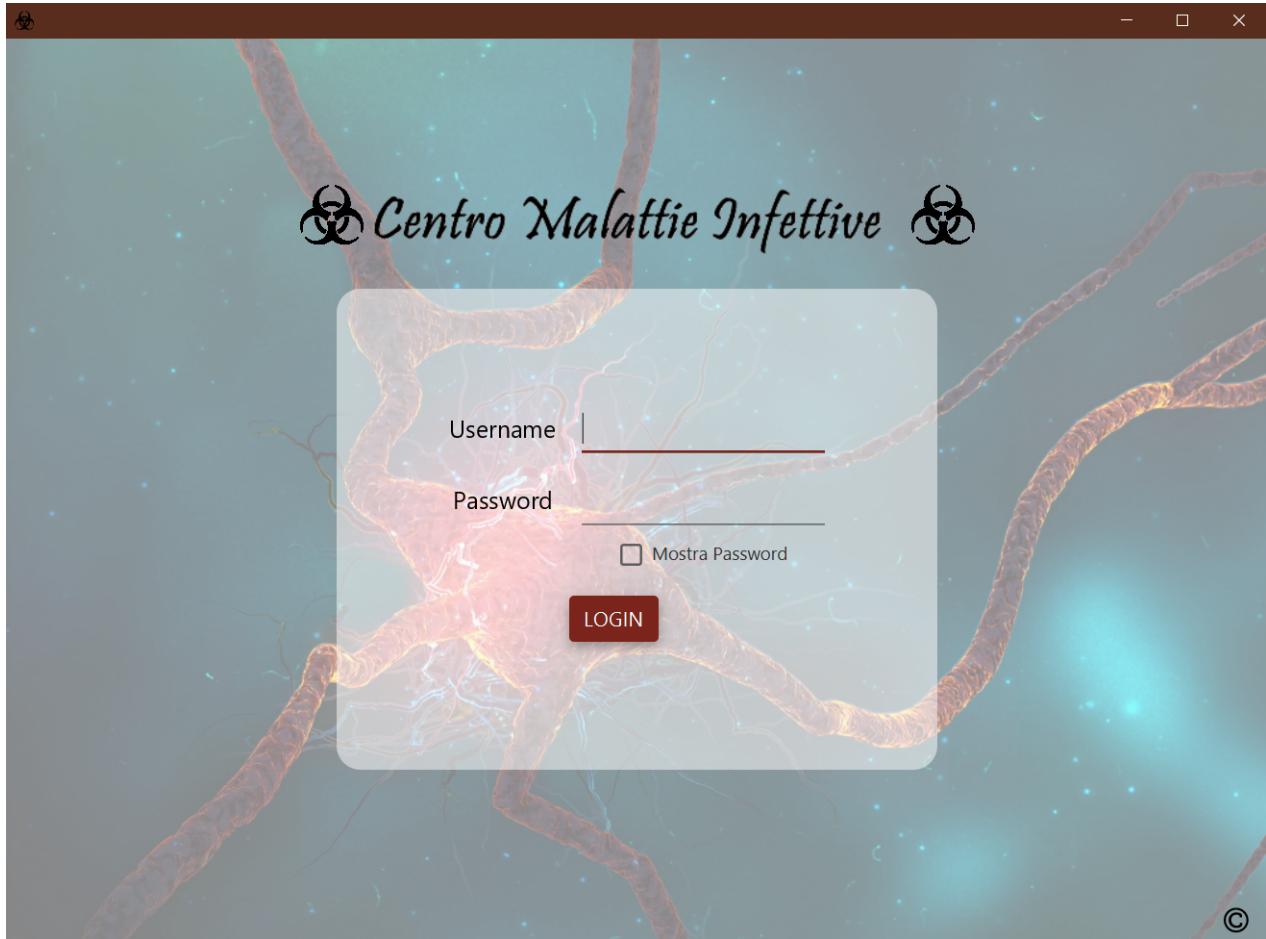
The screenshot shows the Firebase Cloud Firestore interface with the following structure:

- Root level:
 - elaborato-ingegneria
- Under 'elaborato-ingegneria':
 - + Avvia raccolta
 - comuni
 - contagi
 - decessi
 - decessi_malattie_contagiose
 - malattie_contagiose** (highlighted)
 - province
 - regioni
 - users
- Under 'malattie_contagiose':
 - + Aggiungi documento
 - covid** (highlighted)
 - epatiti
 - gastroenteriti
 - meningite
 - morbillo
 - polmonite
 - tubercolosi
- Under 'covid':
 - + Avvia raccolta
 - + Aggiungi campo
 - complications
 - 0 "complicazioni respiratorie"
 - 1 "perdita gusto"
 - 2 "perdita olfatto"
 - name : "covid"

Organizzazione della GUI

Il prototipo è strutturato in circa 2 sezioni principali:

Login page



La prima pagina dell'applicazione è quella di Login: permette l'autenticazione di diversi tipi di utenti (ADMIN, Ricercatori Analisti, Personale dell'ente etc), i quali, una volta loggati, potranno avere accesso alle funzioni che spetta ai loro ruoli. C'è la possibilità inoltre, di registrare un nuovo utente con un determinato ruolo.

Per migliorare la sicurezza è stata aggiunta una criptazione della password in fase di registrazione, in modo tale da non avere nessuna password in chiaro su database La libreria che è stata usata per fare ciò è: **com.google.guava:guava:18.0**

Dashboard

In base al ruolo dell'utente che effettua il login, verranno caricate dinamicamente le voci di menu nella dashboard a seconda dei privilegi:

- **PERSONALE MONITORAGGIO**
 - Inserimento/modifica Regione
 - Inserimento/modifica Provincia
 - Inserimento/modifica Comune
- **PERSONALE CONTAGI**
 - Inserimento settimanale contagi dei comuni di competenza
- **PERSONALE DECESSI**
 - Inserimento annuale decessi per provincia
 - Report decessi
- **RICERCATORE ANALISTA**

- Report contagi e malattie contagiose
- Report decessi
- ADMIN
 - Visibilità su tutti i moduli dell'applicazione

Di seguito ci sono le principali viste accessibili dal menù.

Provincia/Regioni/Comuni

View e Edit Province/Regioni/Comuni

ACTI...	NOME	SUPERFICIE	REGIONE
	Bergamo	40.16	Lombardia
	Milano	181.9	Lombardia
	Padova	852.0	Veneto
	Torino	130.2	Piemonte
	Trento	8522.0	Trentino Alto Adige
	Trieste	52.0	Friuli Venezia Giulia
	Venezia	414.16	Veneto
	Verona	206.6	Veneto

I moduli di province/comuni/regioni permettono di aggiungere una nuova regione, provincia o comune attraverso il pulsante 'inserisci'. In questo prototipo è stata inserita anche la funzionalità di modifica di queste informazioni attraverso l'apposito pulsante. Inoltre è stato anche pensato di implementare nella prossima relase la funzionalità di delete di un oggetto.

Utenti

Edit Utente

Nome Aldo Cognome Baglio

Cod. Fiscale LDOBGL89C05L987R Username aldo.baglio

Password Ripeti Password

Ruolo PERSONALE_CONTAGI

Comuni associati all'utente:

Caldiero, Zevio, Sorgà

Nogara
Caldiero
Mantova
Isola della scala
Zevio
Legnago
Sorgà

REP. DECESSI NAZIONI

LOGOUT INFO

Anche il modulo *utenti* segue la stessa logica di aggiunta/modifica, permettendo inoltre di associare e rimuovere i comuni di competenza per ogni personale a contratto. Nella prossima release del prototipo è stato già pensato di implementare anche la modifica della password per gli utenti.

NOTA: un utente può avere comuni associati solo se ricopre il ruolo di Personale Contagi o di ADMIN.

Per i moduli comuni, regioni , province, utenti è stato implementata una funzione di export dati in modo dinamico nei diversi formati: **TXT, CSV, XLS**. E' già stato pianificato anche di aggiungere l'export in **JSON e XML**.

La funzione di export è disponibile inoltre per tutti i report su malattie contagiose e decessi aggregati nei vari livelli di comune, provincia, regione e nazione.

Contagi Comuni & Decessi Province

Queste due sezioni, gestite rispettivamente da Personale Contagi e Personale Decessi, permettono di inserire con cadenza settimale (nel caso dei contagi) o con cadenza annuale (nel caso dei decessi) i dati, e al contempo visualizzarli mediante appositi filtri.

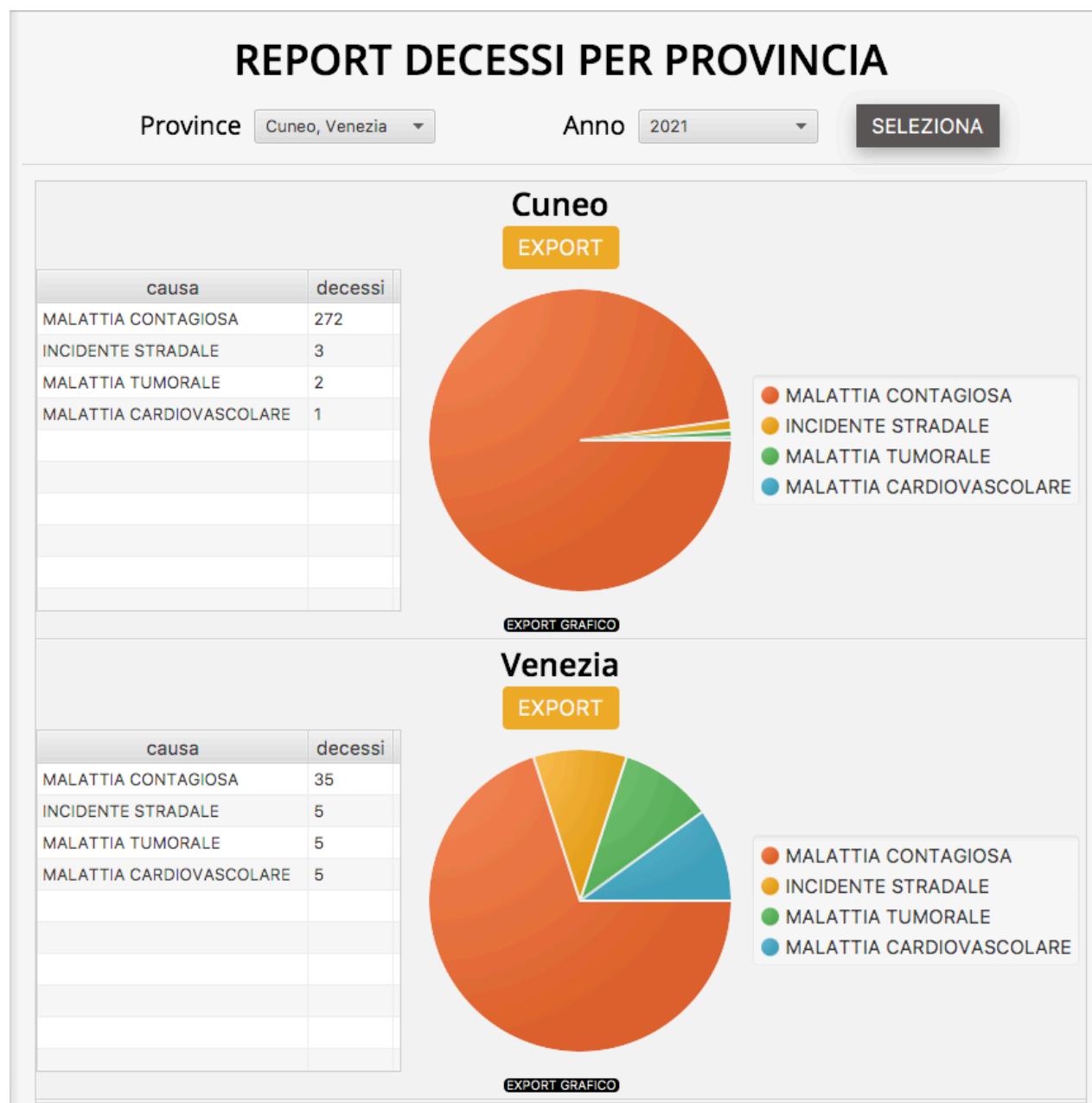
Le **malattie** e le **complicazioni** della sezione contagi comuni vengono **caricate dinamicamente** dal database, permettendo una scalabilità dell'applicazione senza necessità di modifiche/aggiunte direttamente nel codice.

Infatti, ogni malattia contagiosa può avere zero o più complicazioni che vengono salvate nel database, al momento vengono inserite a mano dagli sviluppatori ma in una successiva release del prodotto si potrebbe pensare di mettere a disposizione un modulo apposito per la gestione dinamica delle malattie contagiose da parte di un certo tipo di utenti.

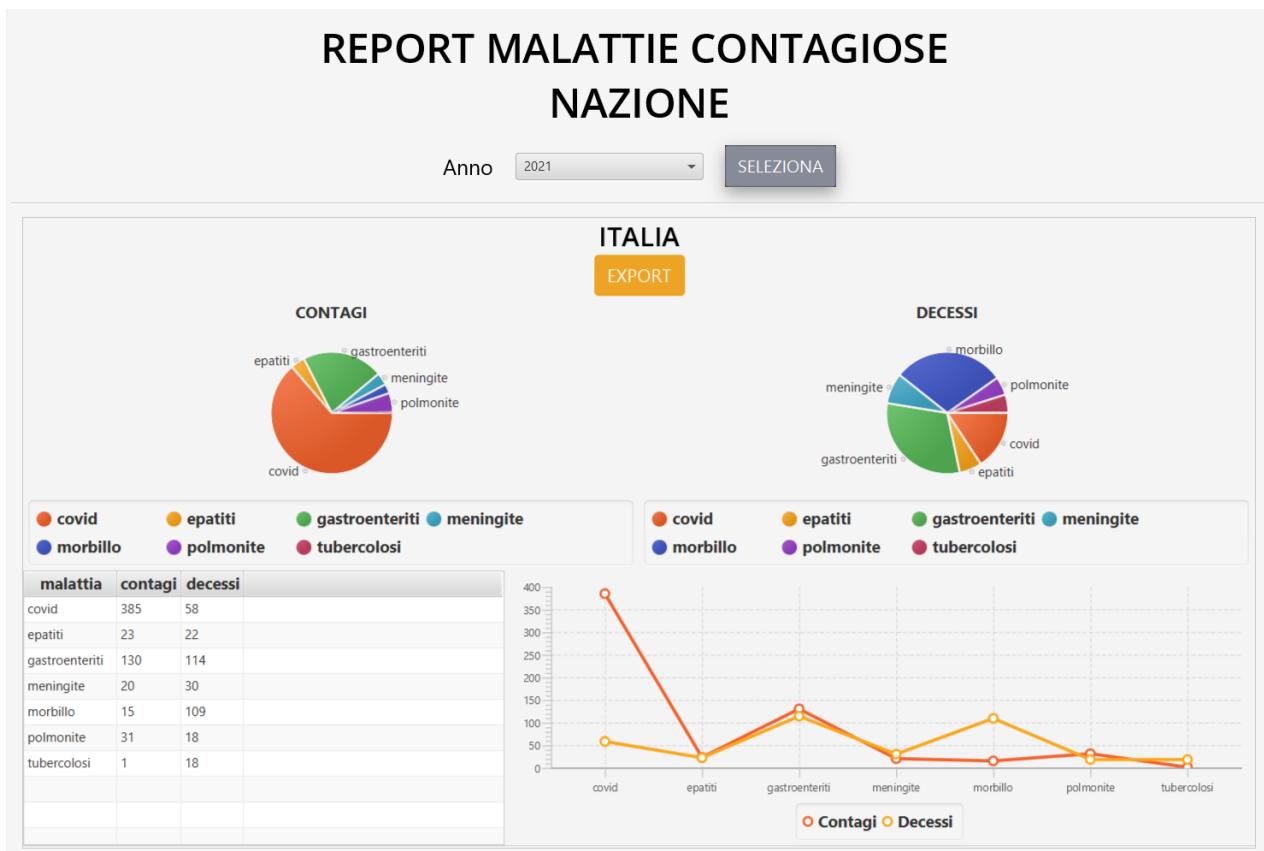
Lo stesso discorso vale anche nei i decessi per provincia, dove le cause per malattie contagiose sono caricate dinamicante.

NOTA: In seguito ad una modifica/aggiunta di un dato nell'applicazione, essa ricarica la pagina, cioè la tabella contenente i dati modificati. Questo per permettere il caricamento dei dati in tempo reale e quindi non dover far refreshare la pagina ogni volta.

Report Decessi



Report Malattie Contagiose



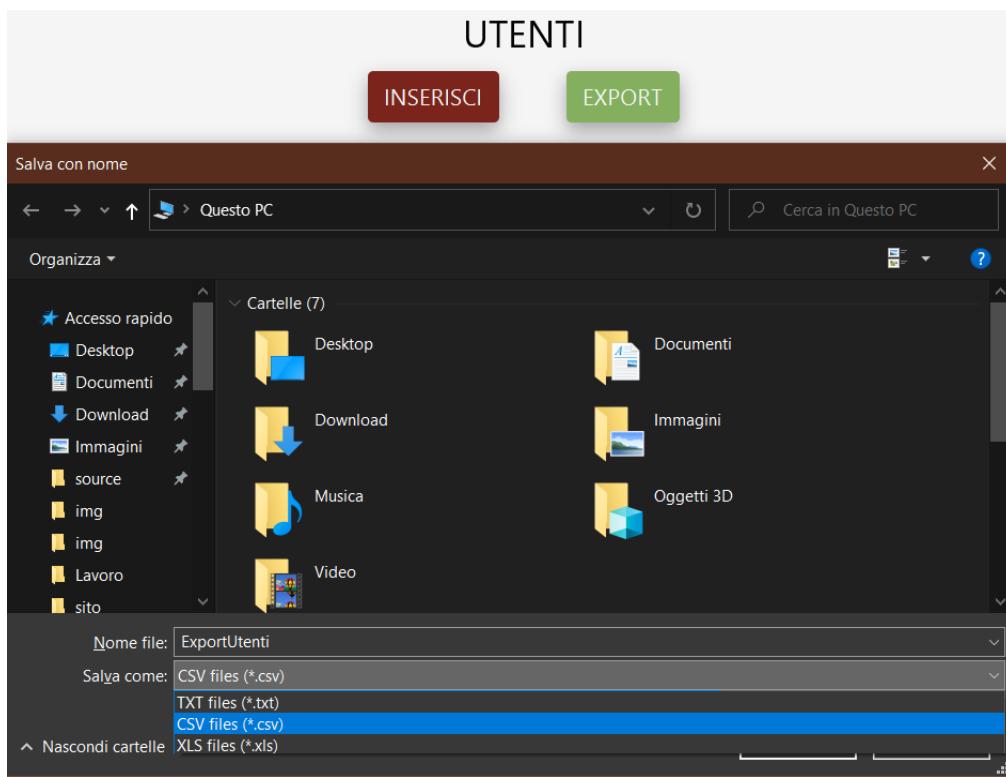
I report per i decessi e delle malattie contagiose sono divisi in tre moduli: per provincia, regione e nazione, oltre al report dei soli contagi per i comuni. Ogni report permette di aggregare e disgregare i dati in base alla tipologia e all'anno. Basandoci e ispirandoci al detto "**un'immagine vale più di mille parole**" è stato deciso di aggiungere svariati report perlopiù grafici (PieChart e LineChart), perchè crediamo che l'impatto visivo sia di gran lunga migliore rispetto all'impatto che può avere una tabella, in quanto più utili e che permettono di imprimere nella memoria il dato. Infine, ultimo ma non per importanza, i grafici risultano essere sicuramente molto utili per avere una visione a colpo d'occhio della situazione generale, mentre i dati nelle tabelle possono tornare utili nel caso si voglia entrare più nel dettaglio e fare una analisi più approfondita.

Inoltre è stata aggiunta la possibilità di esportare tutte le informazioni aggregate presenti nei report, sia sotto forma di tabella, sia sotto forma di charts.

Export

Per i moduli comuni, regioni , province, utenti è stato implementata una funzione di export dati in modo dinamico nei diversi formati: **TXT, CSV, XLS**. E' già stato pianificato anche di aggiungere l'export in **JSON e XML**.

La funzione di export è disponibile inoltre per tutti i report su malattie contagiose e decessi aggregati nei vari livelli di comune, provincia, regione e nazione.



Test e validazione

Introduzione

Per verificare la solidità del software prodotto, si sono svolte le seguenti attività:

1. Ricognizione del documento delle specifiche e confronto con i diagrammi prodotti
2. Verifica della consistenza tra diagrammi e codice prodotto
3. Ispezione del codice, verifica della correttezza dei pattern, ricerca di incogruenze varie
4. Test degli sviluppatori sul software
5. Test utente generico sul software

Ispezione codice e documentazione

In questa fase si è rivisto il documento delle specifiche e lo si è confrontato con i diagrammi UML prodotti, per verificare la correttezza degli use case, activity diagrams e diagramma delle classi. Una volta finita questa attività si è confrontato il codice (staticamente) ai diagrammi UML, per verificarne la consistenza. Infine si è data una nuova ispezione del codice per cercare infrazioni, incogruenze e cattivi usi dei pattern.

Test degli sviluppatori

In questa fase gli sviluppatori hanno immesso nel sistema degli input (sia corretti sia errati) per vedere se la reazione del software fosse quella attesa. I principali test svolti sono:

- **Verifica del corretto funzionamento dell'autenticazione:** i dati errati vengono respinti, ed a fronte dei dati corretti all'utente viene mostrata la schermata iniziale legata alla tipologia di utenza, caricando dinamicamente le corrette informazioni visibili in base al ruolo utente.

- **Verifica del corretto funzionamento della registrazione/modifica di una regione:** se i campi non sono completati nel modo corretto viene segnalato un errore tramite un alert dialog. Controllo anche per la corretta immissione della superficie.
- **Verifica del corretto funzionamento della registrazione/modifica di una provincia:** se i campi non sono completati nel modo corretto viene segnalato un errore tramite un alert dialog. Controllo anche per la corretta immissione della superficie e la corretta selezione della regione di appartenenza.
- **Verifica del corretto funzionamento della registrazione/modifica di un comune:** se i campi non sono completati nel modo corretto viene segnalato un errore tramite un alert dialog. Controllo anche per la corretta immissione della superficie e la corretta selezione della regione di appartenenza. Aggiunto un ulteriore controllo per l'immissione di un corretto *codice ISTAT* per il comune attraverso regex:
 - `Pattern.compile("[0-9]{6}", Pattern.CASE_INSENSITIVE);`
- **Verifica del corretto funzionamento della registrazione/modifica di un utente:** se i campi non sono completati nel modo corretto viene segnalato un errore tramite un alert dialog. Controllo anche per la corretta selezione del ruolo di appartenenza. Aggiunto un ulteriore controllo per l'immissione di un corretto *codice fiscale* per il comune attraverso regex:
 - `Pattern.compile("[a-zA-Z]{6}[0-9]{2}[abcdehlmprstABCDEFHLMPRST]{2}([a-zA-Z][0-9]{3})[a-zA-Z]", Pattern.CASE_INSENSITIVE);`

Per ogni sezione sono stati fatti numerosi test inserendo dati non omogenei, lasciando campi vuoti, di tipologia errata, non congrui per verificare il comportamento dell'applicazione e il sollevamento di eventuali errori o eccezioni. Verificate eventuali risposte relative all'errore creato.

Test esterno

Come ultimo test abbiamo deciso di far usare il prototipo a due persone esterne dallo sviluppo in modo tale da trovare eventuali "punti deboli" dell'applicazione e capire meglio quali sono eventuali aspetti desiderati degli utenti comuni in fase di uso, che il team di sviluppo non aveva ancora pensato.

In questa fase non si è cercato in nessun modo di guidare o strutturare l'esperienza, per non influenzare in alcun modo il risultato; piuttosto si è lasciato che il soggetto navigasse liberamente il sistema.

Non è stata data nessuna spiegazione sull'utilizzo del software, se non una generale indicazione dei suoi fini; ci si è limitati a rispondere alle domande, quando sollevate.

L'unico scopo del test era quello di rilevare errori invisibili allo sviluppatore; in realtà, gli utenti a cui è stato mostrato il software hanno anche aiutato ad individuare nuove funzionalità per migliorare l'usabilità generale del sistema. Forse quest'ultimo test è stato uno dei più utili in quanto è stato talvolta fonte di idee e di discussione tra il team, dando un'idea anche di quali parti siano *user-friendly* e quali no.