System Design

Introduzione

In questo documento analizzeremo l'architettura software, System Design e Object Design, e descrivermo i Design Pattern utilizzati, tutto rappresentato con diagrammi UML 2.0 (ClassDiagram, e SequenceDiagram).

Architettura Software

Per l'Architettura Software, abbiamo deciso di usare una architettura *Three-Tier,* utilizzando il Design Pattern **MVC**, per rappresentare i 3 livelli del software, questa scelta ci ha portato ad avere una flessibilità e modificabilità maggiore del sistema, poiché avendo più livelli, dove ogni livello svolge un compito ben preciso, e nessun altro, possiamo facilmente individuare i vari bug, e quindi modificare facilmente una componente senza che le altre vengano modificate di conseguenza, da qui deriva una potenziale riusabilità di componenti per sistemi diversi, i livello sono i seguenti:

- **Livello di presentazione:** Sono tutte le UI del sistema, le interazioni con l'utente, e come vengono rappresentate le informazioni.
- **Livello di Business:** In questo livello abbiamo tutte le regole di business, quindi il cuore dell'applicazione, controllerà i flussi di dato dal livello di presentazione e il data layer.
- **Livello dei Dati:** Qui invece troveremo tutte quelle classi che interagiranno con i dati veri e propri dell'applicazione, quindi con la nostra base di dati scelta.

Come si può notare il '*Livello di presentazione'* e il *'Livello dei Dati'* non sapranno la reale implementazione e neanche la loro ' esistenza ', per cui delle modifiche a uno dei due livelli non toccherà l'altro, e viceversa, mentre un discorso diverso per il *'Livello di Business'*, che dovendo regolare il flusso dei dati tra i due livello, verrà influenzato anche se di poco dai vari cambiamenti dei due livello.

Usando questo tipo di architettura, avremo un codice molto voluminoso, rinunciando anche in parte alla performance del sistema, preferendo una maggiore facilità di manutenzione e favorendo i futuri cambiamenti, relativamente ai cambiamenti, abbiamo avuto un'esperienza diretta, trovandoci a cambiare durante questa fase del progetto, a cambiare spesso implementazione e design , i cambiamento sono stati molto rapidi, ed hanno coinvolto poche classi del sistema.

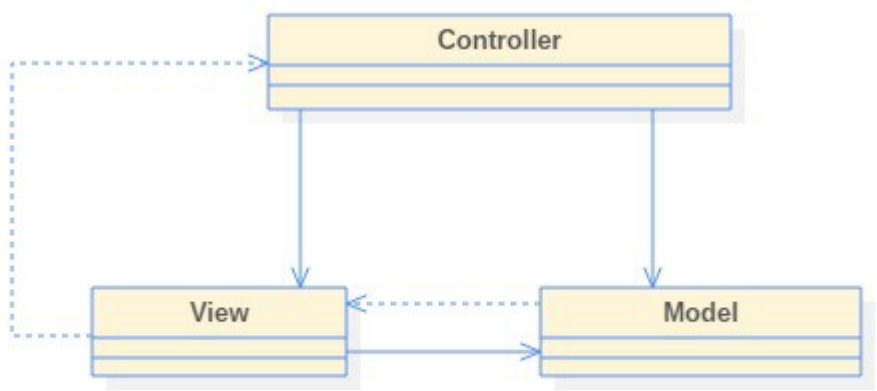
Tecnologie Utilizzate:

- **JavaFX:** Software basato su Java, per Rich Internet Application, ci è tornato molto utile per la modellazione della parte grafica, utilizzando file FXML per la modellare la UI, e alla possibilità di integrare CSS al suo interno, e la possibilità di importare numerose librerie grafiche.
- **Java 8:** linguaggio di programmazione orientato agli oggetti utilizzato.
- **Oracle Database Express Edition 11g:** Questo è il nostro DBMS scelto per la memorizzazione dei dati nella nostra applicazione, lo abbiamo scelto per i vari servizi che offre Oracle e la grande compatibilità con Java.
- **SceneBuilder:** Programma di modellazione grafica per JavaFX, rendendo la modellazione UI molto intuitiva e semplice, lasciandoci più spazio per l'implementazione e la fase di testing del software.

- **Jfoenix:** Librearia grafica per JavaFX, la quale implementata Material Design di Google: https://github.com/jfoenixadmin/JFoenix
- **GitHub**: E' stata la nostra scelta per il software di versioning, ci ha aiutato molto nel condividere e analizzare il codice degli altri componenti del gruppo. ( )https://github.com/JimMinor/EventManager2018

In Dettaglio, Design Pattern e Best Practices:

Abbiamo utilizzando in gran parte il **pattern architetturale MVC:**

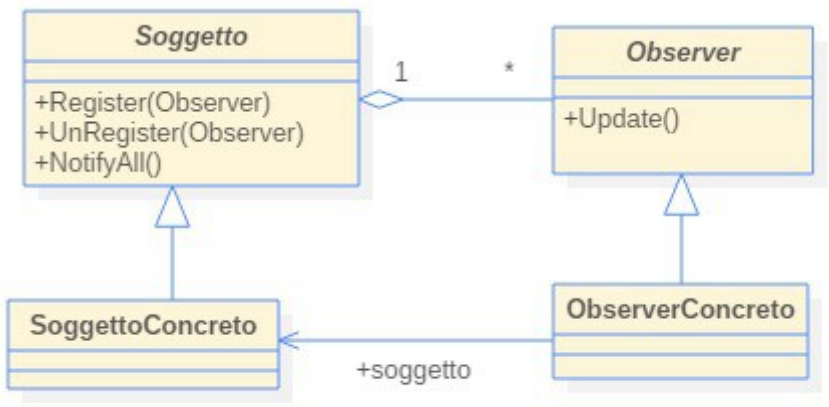


Per implentare parte della nostra architettura, sopratutto per quanto riguarda, i primi due livelli :

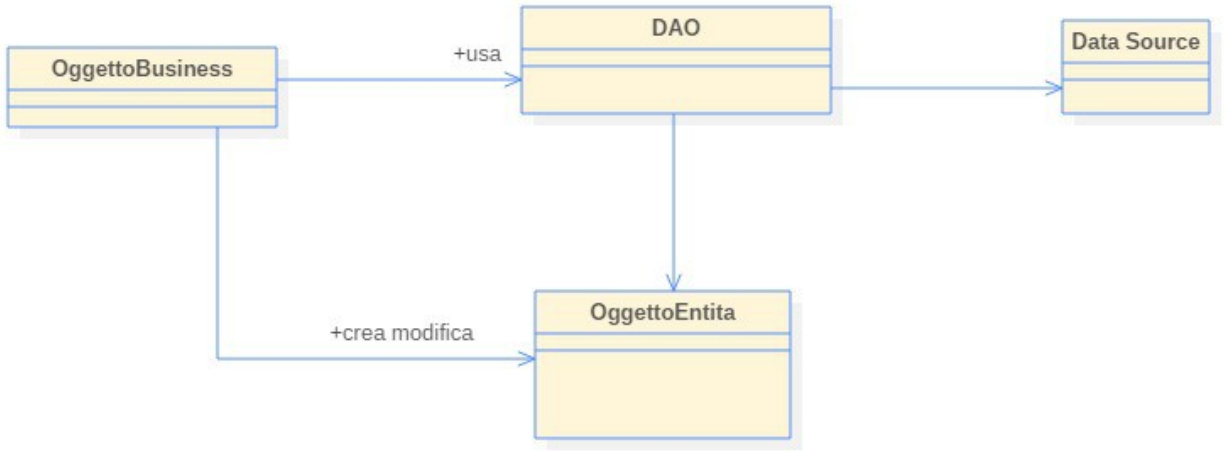
- **Presentazione**
- **Business**

Questo ci ha portato a molti vantaggi, quali estendibilità, riusabilità, modalurità, e manutenibilità.

Come punto di partenza per implementare molte delle funzionalità del sistema, adattandolo alle tecnologie utilizzate.
Sopratutto l'implementazione della **View,** la quale è stata implementata con la consueta best practice, seguendo il Design Pattern **Obsever**.



Invece per il **Livello dei Dati**, abbiamo usato il Design Pattern **DAO**, il quale ci permetterà una maggiore facilità di manutenzione in caso di cambiamenti sulla basi di dati, e favorendo più implementazioni di basi di dati differenti, e diminuisce l'accoppiamento.



Inoltre, abbiamo fatto uso della **Legge di Demetra**, per avere un *Basso Accoppiamento* tra le classi.

Dal Dominio all' Implementazione:

Una volta steso un documento del modello Funzionale e di Dominio, siamo entrati sempre nel più nel dettaglio del sistema, e implementando le funzionalità in base alle tecnologie scelte, per questo motivo abbiamo dovuto modificare alcuni Design Pattern.

MVC e JavaFX
JavaFX introduce i file FXML, e altre molte funzionalità, permettendo di avere una classe che controllerà il file FXML (che ne descriverà l'aspetto della UI), questa classe potrà essere sia come vista

che come controller, ma questa scelta non ci è sembrata la più consona per il nostro obiettivo, per questo motivo abbiamo deciso di rimanere più vicino alla 'classica' implementazione del pattern **MVC**, abbiamo dunque utilizzato un file FXML, e abbiamo associato a quest'ultimo una classe con i soli riferimenti agli oggetti grafici, quali Button e TextField etc, impostando questa classe come controller della risorsa FXML, successivamente abbiamo creato un vero e proprio controller, il quale al suo interno avrà la **Business Logic**, e i Listener per ogni Component della View, in modo tale da avere tutta la logica in unica classe, che interagirà anche con gli oggetti di tipo **DAO** e **Entity**. Dove necessario la **View** implementarà **Observer**, e il **Model** sarà l'oggetto **Observable**, e il **Controller** potrà essere visto come un **ActionListener**.

Data Layer e DAO:
Per implementare questo Layer, abbiamo si usato le classi DAO implementandole secondo il basi di dati utilizzata, ma queste classi sono state utilizzate solo per gestire gli errori (*SQLException)* lanciate dalle funzioni che comunicano direttamente con la base di dati.
Le query sono state gestite da classi apposite, perché abbiamo notato che alcune query potevano essere usate da più oggetti DAO che gestivano entità diverse, così da rendere le classi più leggibili e più facile di mantenere, mantenendo un basso accoppiamento.

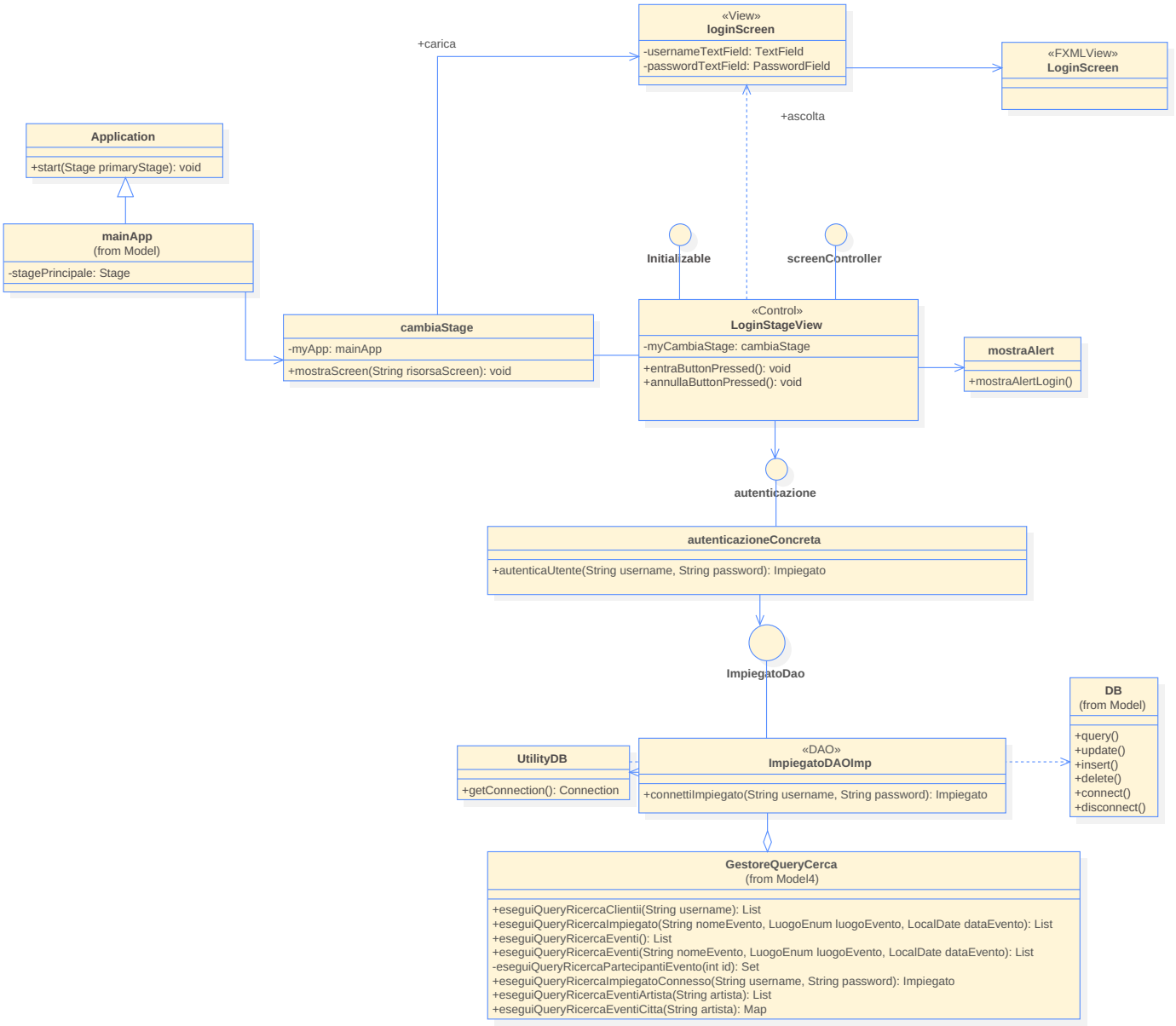
Proseguiamo con la presentazione dei diagrammi **UML**:

Login:

Questa funzionalità è stata aggiunta in questa fase, un Impiegato o un Amministratore posso entrare nell'applicazione con le credenziali scelte nella parte Front-End della piattaforma, questo permetterà di 'chiudere' le aree riservate agli Impiegati.

E' stata usata un interfaccia *Autenticazione* in modo da facilitare futuri cambiamenti di autenticazione dell utente o di aggiungerne altri a seconda delle necessità, è stata implementata un autenticazione con Username e Password.

E' stata usata di un interfaccia per l'Autenticazione, in caso si voglia implementare un altro metodo o aggiungere più metodi di autenticazione diversi.


«View»
loginScreen
-usernameTextField: TextField
-passwordTextField: PasswordField
+carica
«FXMLView»
LoginScreen
+ascolta
Application
+start(Stage primaryStage): void
mainApp
(from Model)
-stagePrincipale: Stage
Initializable
screenController
cambiaStage
-myApp: mainApp
+mostraScreen(String risorsaScreen): void
«Control»
LoginStageView
-myCambiaStage: cambiaStage
+entraButtonPressed(): void
+annullaButtonPressed(): void
mostraAlert
+mostraAlertLogin()
autenticazione
autenticazioneConcreta
+autenticaUtente(String username, String password): Impiegato
ImpiegatoDao
DB
(from Model)
+query()
+update()
+insert()
+delete()
+connect()
+disconnect()
UtilityDB
+getConnection(): Connection
«DAO»
ImpiegatoDAOImp
+connettiImpiegato(String username, String password): Impiegato
GestoreQueryCerca
(from Model4)
+eseguiQueryRicercaClientii(String username): List
+eseguiQueryRicercaImpiegato(String nomeEvento, LuogoEnum luogoEvento, LocalDate dataEvento): List
+eseguiQueryRicercaEventi(): List
+eseguiQueryRicercaEventi(String nomeEvento, LuogoEnum luogoEvento, LocalDate dataEvento): List
-eseguiQueryRicercaPartecipantiEvento(int id): Set
+eseguiQueryRicercaImpiegatoConnesso(String username, String password): Impiegato
+eseguiQueryRicercaEventiArtista(String artista): List
+eseguiQueryRicercaEventiCitta(String artista): Map

Login CRC CARD

Nome Classe: *ImpiegatoDAOImp*	**Package**: DB
Superclasses	*ImpiegatoDAO*
Subclasses	-
Responsabilità	**Collaboratori**
Operazioni di C.R.U.D sulla classe *Impiegato*	*GestoreQueryCerca*

Nome Classe: LoginCommand	**Package**:
Superclasses	Command
Subclasses	-
Responsabilità E' un costruttore che contiene username e password e serve per il login all applicativo	**Collaboratori**

Nome Classe: Impiegato	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Genera un costruttore per creare un nuovo Impiegato	**Collaboratori**

Nome Classe: VisualizzaImpiegatoModel	**Package**: Model
Superclasses	Observable
Subclasses	-
Responsabilità Creare un setlist per gli impiegati e un oggetto per l'impiegato selezionato	**Collaboratori**

Nome Classe: *LoginStage*	**Package**: View
Superclasses	Initizializable,ControlledScreen
Subclasses	-
Responsabilità	**Collaboratori**
Questa classe si occupa di rappresentare la schermata di login e, controllare la correttezza dei dati inseriti per il login, e di ritornare l'istanza 'Impiegato' corrispondente .	*CambiaStage,Autenticazione,Impiegato*

Gestione Eventi:

La Gestione degli Eventi comprende tutte le funzionalità descritte nel documento di Dominio è sono:

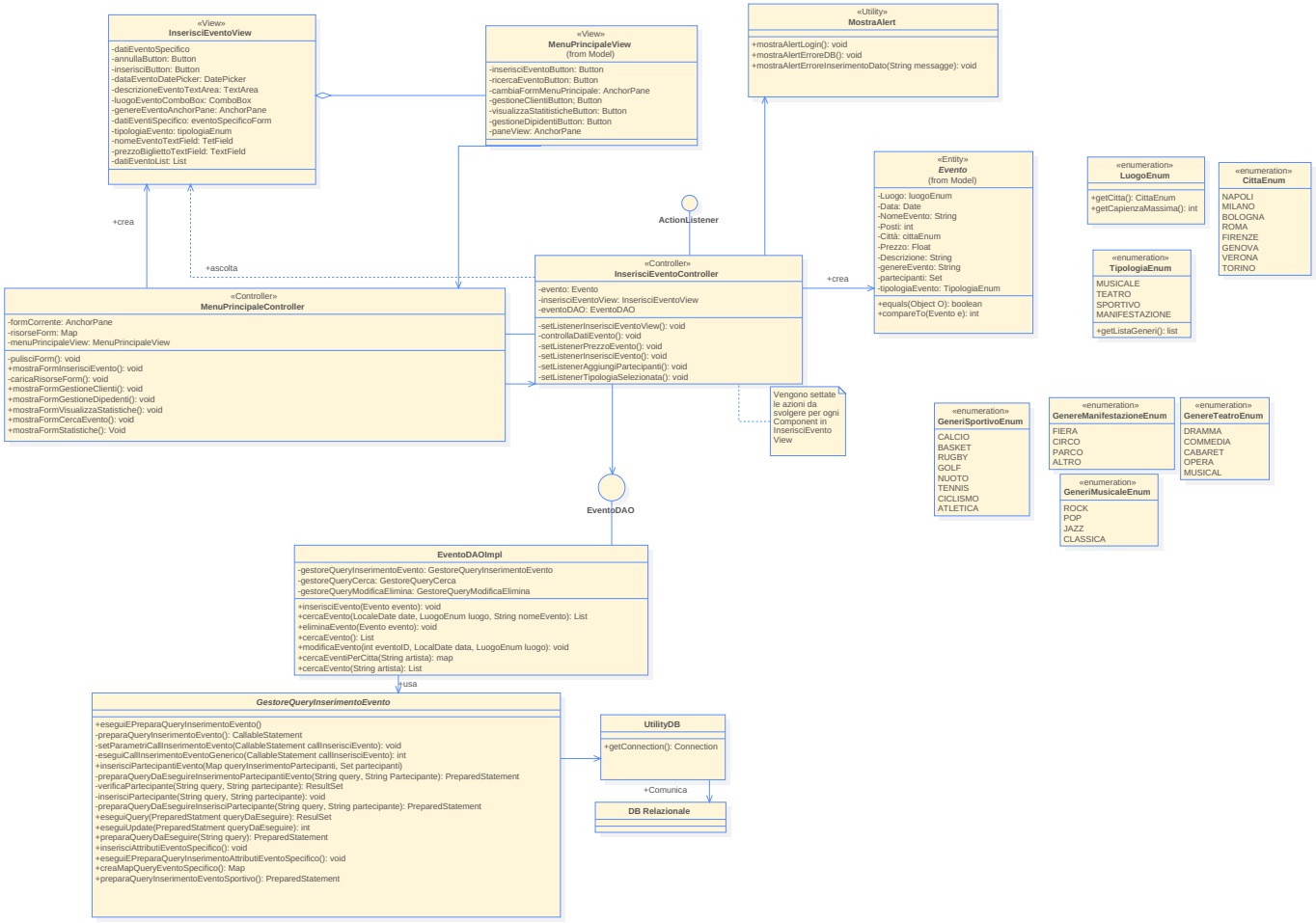
- Inserimento Evento
- Ricerca Evento
- Cancellazione Evento
- Modifica Evento
- Visualizzare Evento

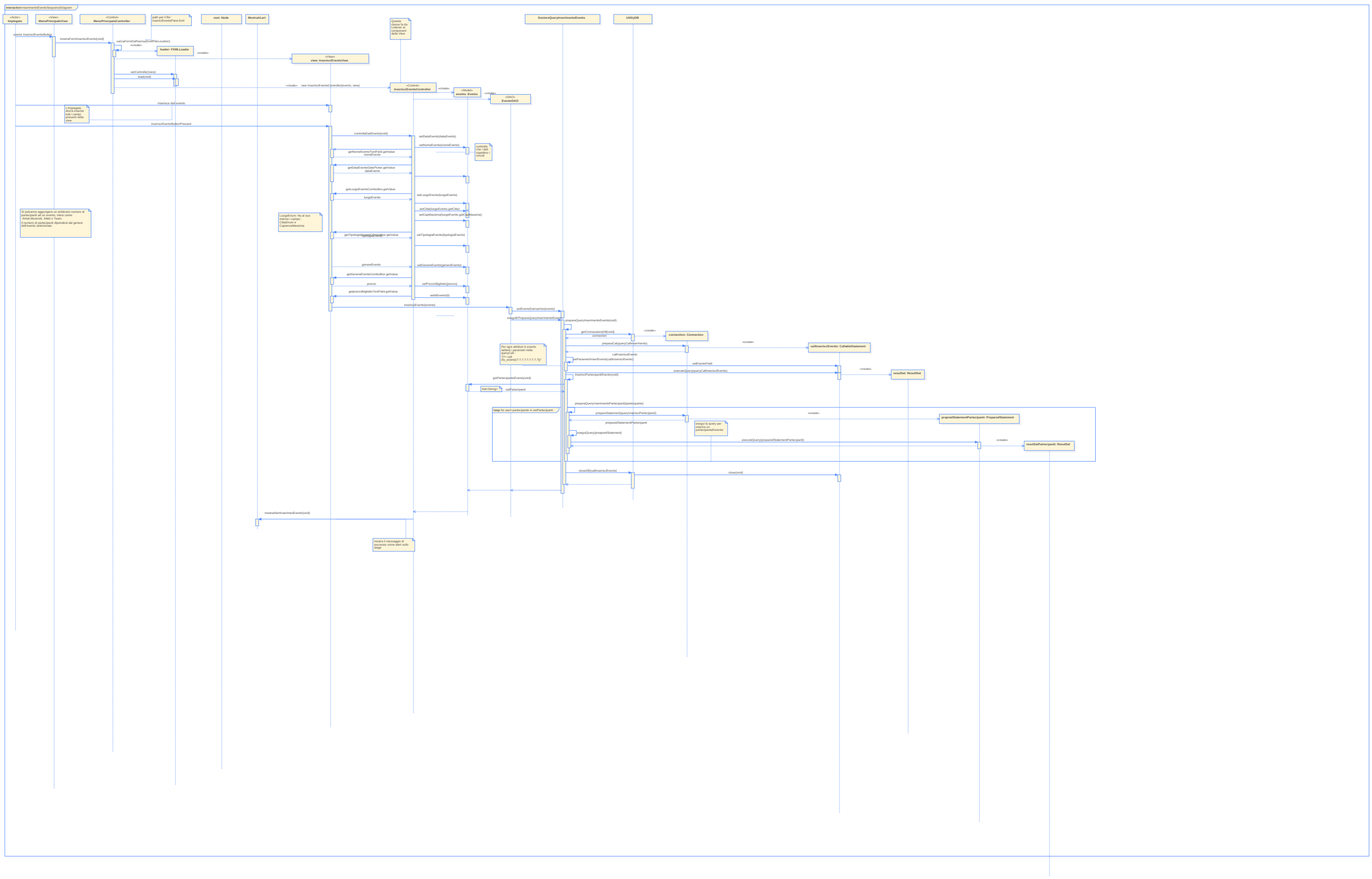
Inserimento Evento

In questa funzionalità non è stato implementato un vero e proprio MVC, ma è stato usato come Model l'entita Evento, la quale verrà creata dal controller che 'ascolterà' le azioni dei Componet nella View, alle quali verrà associata una funzione o un controllo sui dati al verificarsi di un evento.

La View ha i riferimenti agli oggetti del form, il come verranno presentati è descritto nel file FXML della View.

Il MenuPrincipaleController si occuperà di creare tutti gli oggetti utili completare la funzionalità dell inserimento di un Evento, inserirà nell'AnchorPane nella vista MenuPrincipaleView il form per la creazione di un evento InserisciEventoView, avrà al suo interno tutti i Component necessari per instanziare un entità di tipo Evento.
Sono state usate in gran quantità le Enum, per facilitare i controlli sui dati che dovranno essere inseriti, i controlli sui dati effettivi sono diventati pochi, così da diminuire i casi di errore.
Verrà dunque controllato che nell TextField del prezzo di un biglietto, verranno inseriti solo Float, (grazie ad una REGEXP), l'uso delle ComboBox per la Tipologia di un evento e il suo Genere, non porterà ad incompatibilità tra Tipologia e i suoi sottogeneri, in quanto solo una volta selezionata la tipologia la ComboBox per il genere verrà 'riempita' dalla lista di sottogeneri associati a quella specifica tipologia di evento selezionata.
I partecipanti all'evento non potranno avere duplicati, poiché abbiamo sfruttato le proprietà dei Set in Java i quali non prevedono duplicati al suo interno, in questo modo abbiamo avuto bisogno di effettuare ulteriori controlli.


«View»
InserisciEventoView
-datiEventoSpecifico
-annullaButton: Button
-inserisciButton: Button
-dataEventoDatePicker: DatePicker
-descrizioneEventoTextArea: TextArea
-luogoEventoComboBox: ComboBox
-genereEventoAnchorPane: AnchorPane
-datiEventiSpecifici: eventoSpecificoForm
-tipologiaEvento: tipologiaEnum
-nomeEventoTextField: TetField
-prezzoBiglettoTextField: TextField
-datiEventoList: List
«View»
MenuPrincipaleView
(from Model)
-inserisciEventoButton: Button
-ricercaEventoButton: Button
-cambiaFormMenuPrincipale: AnchorPane
-gestioneClientiButton; Button
-visualizzaStatitisticheButton: Button
-gestioneDipidentiButton: Button
-paneView: AnchorPane
«Utility»
MostraAlert
+mostraAlertLogin(): void
+mostraAlertErroreDB(): void
+mostraAlertErroreInseimentoDato(String messagge): void
+crea
+ascolta
ActionListener
«Entity»
Evento
(from Model)
-Luogo: luogoEnum
-Data: Date
-NomeEvento: String
-Posti: int
-Città: cittaEnum
-Prezzo: Float
-Descrizione: String
-genereEvento: String
-partecipanti: Set
-tipologiaEvento: TipologiaEnum
+equals(Object O): boolean
+compareTo(Evento e): int
«enumeration»
LuogoEnum
+getCitta(): CittaEnum
+getCapienzaMassima(): int
«enumeration»
CittaEnum
NAPOLI
MILANO
BOLOGNA
ROMA
FIRENZE
GENOVA
VERONA
TORINO
«enumeration»
TipologiaEnum
MUSICALE
TEATRO
SPORTIVO
MANIFESTAZIONE
+getListaGeneri(): list
«Controller»
MenuPrincipaleController
-formCorrente: AnchorPane
-risorseForm: Map
-menuPrincipaleView: MenuPrincipaleView
-puliciForm(): void
+mostraFormInserisciEvento(): void
-caricaRisorseForm(): void
+mostraFormGestioneClienti(): void
+mostraFormGestioneDipedenti(): void
+mostraFormVisualizzaStatistiche(): void
+mostraFormCercaEvento(): void
+mostraFormStatistiche(): Void
«Controller»
InserisciEventoController
-evento: Evento
-inserisciEventoView: InserisciEventoView
-eventoDAO: EventoDAO
-setListenerInserisciEventoView(): void
-controllaDatiEvento(): void
-setListenerPrezzoEvento(): void
-setListenerInserisciEvento(): void
-setListenerAggiungiPartecipanti(): void
-setListenerTipologiaSelezionata(): void
+crea
Vengono settate le azioni da svolgere per ogni Component in InserisciEvento View
«enumeration»
GeneriSportivoEnum
CALCIO
BASKET
RUGBY
GOLF
NUOTO
TENNIS
CICLISMO
ATLETICA
«enumeration»
GenereManifestazioneEnum
FIERA
CIRCO
PARCO
ALTRO
«enumeration»
GenereTeatroEnum
DRAMMA
COMMEDIA
CABARET
OPERA
MUSICAL
«enumeration»
GeneriMusicaleEnum
ROCK
POP
JAZZ
CLASSICA
EventoDAO
EventoDAOImpl
-gestoreQueryInserimentoEvento: GestoreQueryInserimentoEvento
-gestoreQueryCerca: GestoreQueryCerca
-gestoreQueryModificaElimina: GestoreQueryModificaElimina
+inserisciEvento(Evento evento): void
+cercaEvento(LocaleDate date, LuogoEnum luogo, String nomeEvento): List
+eliminaEvento(Evento evento): void
+cercaEvento(): List
+modificaEvento(int eventoID, LocalDate data, LuogoEnum luogo): void
+cercaEventiPerCitta(String artista): map
+cercaEvento(String artista): List
+usa
GestoreQueryInserimentoEvento
+eseguiEPreparaQueryInserimentoEvento()
-preparaQueryInserimentoEvento(): CallableStatement
-setParametriCallInserimentoEvento(CallableStatement callInserisciEvento): void
-eseguiCallInserimentoEventoGenerico(CallableStatement callInserisciEvento): int
+inserisciPartecipantiEvento(Map queryInserimentoPartecipanti, Set partecipanti)
-preparaQueryDaEseguireInserimentoPartecipantiEvento(String query, String Partecipante): PreparedStatement
-verificaPartecipante(String query, String partecipante): ResultSet
-inserisciPartecipante(String query, String partecipante): void
-preparaQueryDaEseguireInserisciPartecipante(String query, String partecipante): PreparedStatement
+eseguiQuery(PreparedStatment queryDaEseguire): ResulSet
+eseguiUpdate(PreparedStatment queryDaEseguire): int
+preparaQueryDaEseguire(String query): PreparedStatement
+inserisciAttributiEventoSpecifico(): void
+eseguiEPreparaQueryInserimentoAttributiEventoSpecifico(): void
+creaMapQueryEventoSpecifico(): Map
+preparaQueryInserimentoEventoSportivo(): PreparedStatement
UtilityDB
+getConnection(): Connection
+Comunica
DB Relazionale


interaction InserimentoEventoSequenceDiagram
«Actor» Impiegato
«View» MenuPrincipaleView
«Control» MenuPrincipaleController
path per il file ' inserisciEventoPane.fxml
root: Node
MostraALert
Questa classe fa da Listener ai component della View
GestoreQueryInserimentoEvento
UtilityDB
preme InserisciEventoButton
mostraFormInserisciEvento(void)
caricaFormDaRisorsa(fxmlFileLocation)
«create»
loader: FXMLLoader
«create»
«View» view: InserisciEventoView
setController(view)
load(void)
«create» new InserisciEventoController(evento, view)
«Control» InserisciEventoController
«create»
«Model» evento: Evento
«create»
«DAO» EventoDAO
inserisce datiEvento
L'Impiegato dovrà inserire tutti i campi presenti nella view
inserisciEventoButtonPressed
controllaDatiEvento(void)
setDataEvento(dataEvento)
setNomeEvento(nomeEvento)
controlla che i dati rispettino i vincoli
getNomeEventoTextField.getValue
nomeEvento
getDataEventoDatePicker.getValue
dataEvento
getLuogoEventoComboBox.getValue
setLuogoEvento(luogoEvento)
luogoEvento
setCitta(luogoEvento.getCitta)
setCapMassima(luogoEvento.getCapMassima)
Si potranno aggiungere un arbitrario numero di partecipanti ad un evento, intesi come: Artisti,Musicisti, Atleti o Team.
Il numero di partecipanti dipenderà dal genere dell'evento selezionato.
LuogoEnum: Ha al suo interno i campo : CittaEnum e CapienzaMassima
getTipologiaEventoComboBox.getValue
setTipologiaEvento(tipologiaEvento)
genereEvento
setGenereEvento(genereEvento)
getGenereEventoComboBox.getValue
prezzo
setPrezzoBiglietto(prezzo)
getprezzoBigliettoTextField.getValue
setIdEvento(0)
inserisciEvento(evento)
setEventoDaInserire(evento)
eseguiEPreparaQueryInserimentoEvento
preparaQueryInserimentoEvento(void)
getConnessioneDB(void)
connection
«create»
connection: Connection
preparaCall(queryCallInserimento)
«create»
callInserisciEvento: CallableStatement
Per ogni attributi in evento setterà i parametri nella queryCall : "{?= call ins_evento(?,?,?,?,?,?,?,?,?)}"
callInserisciEvento
setParametriInserirEvento(callInserisciEvento)
setEventoField
executeQuery(queryCallInserisciEvento)
«create»
resultSet: ResultSet
inserisciPartecipantiEvento(void)
getPartecipantiEvento(void)
Set<String>
setPartecipanti
preparaQueryInserimentoPartecipanti(partecipante)
loop for each partecipante in setPartecipanti
prepareStatement(queryInserisciPartecipanti)
«create»
prapredStatementPartecipanti: PreparedStatement
preparedStatementPartecipanti
esegui la query per inserire un partecipanteall'evento
eseguiQuery(preparedStatement)
executeQuery(preparedStatementPartecipanti)
«create»
resultSetPartecipanti: ResultSet
closeDB(callInserisciEvento)
close(void)
mostraAlertInserimeoEvento(void)
mostra il messaggio di successo come alert sullo stage

Inserimento Evento CRC Card:
Elencate le specifiche classi per inserimento di un evento.

Nome Classe: LuogoEnum	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Specifica il luogo di un evento definendo il nome della città e la capienza massima	**Collaboratori**

Nome Classe: *InserisciEventoController*	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge Listener ed EventHandler ai component della View, controllando la logica dell inserimento di un evento, controllando che i vincoli di un evento siano stati rispettati, creare un Evento.	**Collaboratori** *MostraAlert, EventoDAO, InserisciEventoView,Evento*

Nome Classe: InserisciEventoView	**Package**: View
Superclasses	AnchorPane
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata di inserimento di un evento Fornendo i getter per i vari componenti della view	**Collaboratori** TipologiaEnum MenuPrincipaleController

Nome Classe: GestoreQueryInserimentoEvento	**Package**: DB
Superclasses	-
Subclasses	-
Responsabilità	**Collaboratori**
Connessione al DB, prepare ed eseguire query per l'iserimento nel database di un evento e dei partecipanti ,in piu cerca anche se non ci sono collisioni per la creazione di un nuovo evento	*UtilityDB*

Nome Classe: EventoDAOImp	**Package**: DB

Superclasses	EventoDAO
Subclasses	-
Responsabilità	**Collaboratori**
Chiama le classi di GestioneQuery per eseguire le query relative alle Operazioni di C.R.U.D per l'entità *Evento.*	*GestoreQueryCerca,* *GestoreQueryInserimento* *GestoreQueryModificaElimina*

Nome Classe: Evento	**Package**: Model
Superclasses	Comparable
Subclasses	-
Responsabilità Genera un costruttore per gli Eventi i getter e i setter per i vari attributi, Viene inizializzato il numero di biglietti a 0 per concedere l'eliminazione	**Collaboratori**

Nome Classe: GenereTeatroEnum	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Spcifica il Genere teatro di un evento	**Collaboratori**

Nome Classe: GenereManifestazioneEnum	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Spcifica il Genere manifestazione di un evento	**Collaboratori**

Nome Classe: SportEnum	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Spcifica il Genere Sport di un evento	**Collaboratori**

Nome Classe: TipologiaEnum	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Assegna ad ogni tipo di evento la lista della sottocategoria	**Collaboratori**

Ricerca Evento:

Questa funzionalità è necessaria per svolgere le funzionalità : **Modifica,Eliminazione,Visualizzazione.**
Per la ricerca di un evento è stato invece utilizzato il Design Pattern **MVC** nel senso più completo, è stato usato un Model che avrà come attributi la Lista di Eventi trovati e un Evento selezionato che servirà per svolgere le altre funzionalità relative alla ricerca.

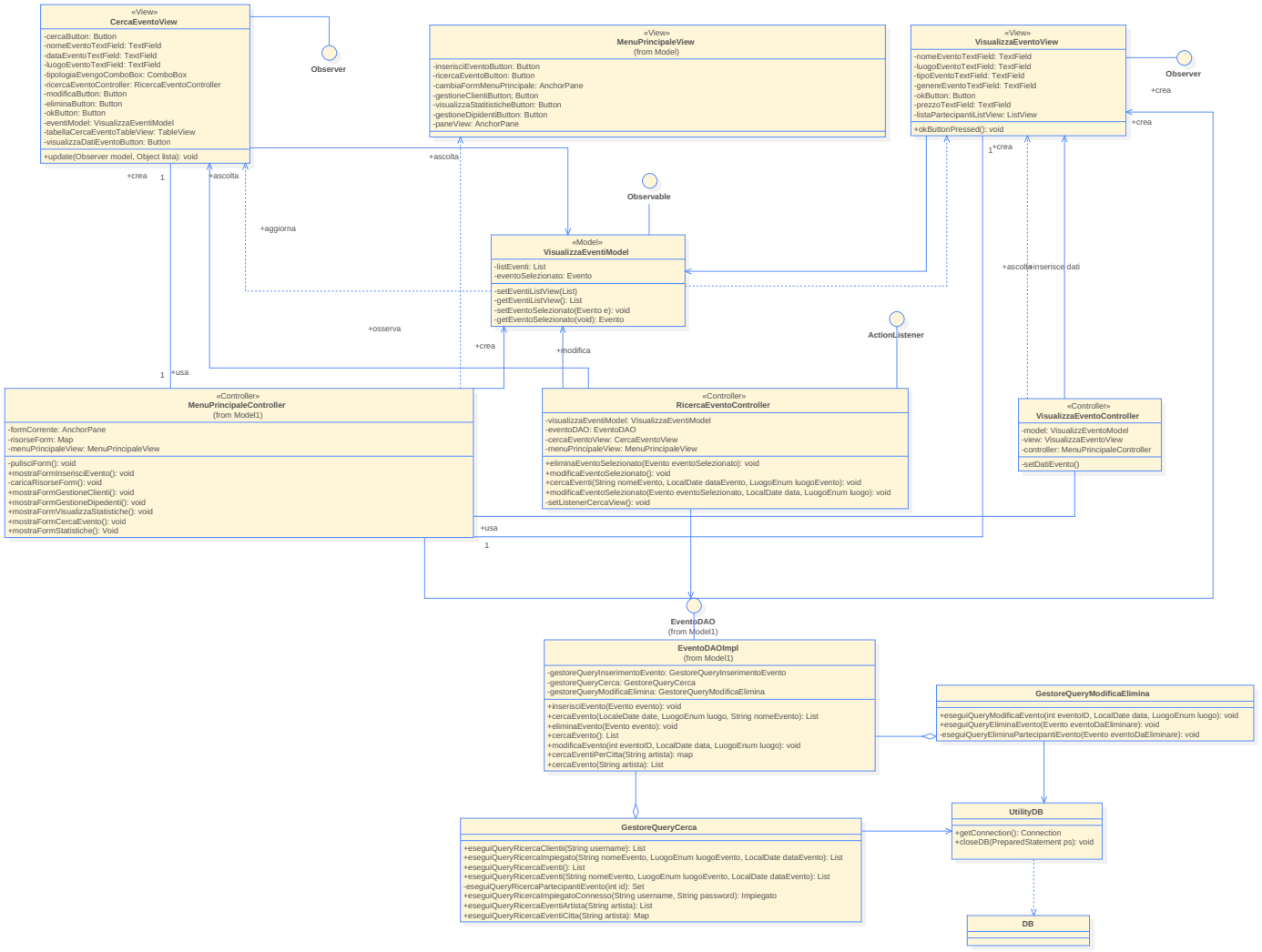
Una volta che sarà premuto il tasto delle ricerca di un evento, verrà effettuata una ricerca di tutti gli Eventi presenti, successivamente si potrà decidere dei parametri per effettuare una ricerca più specifica.

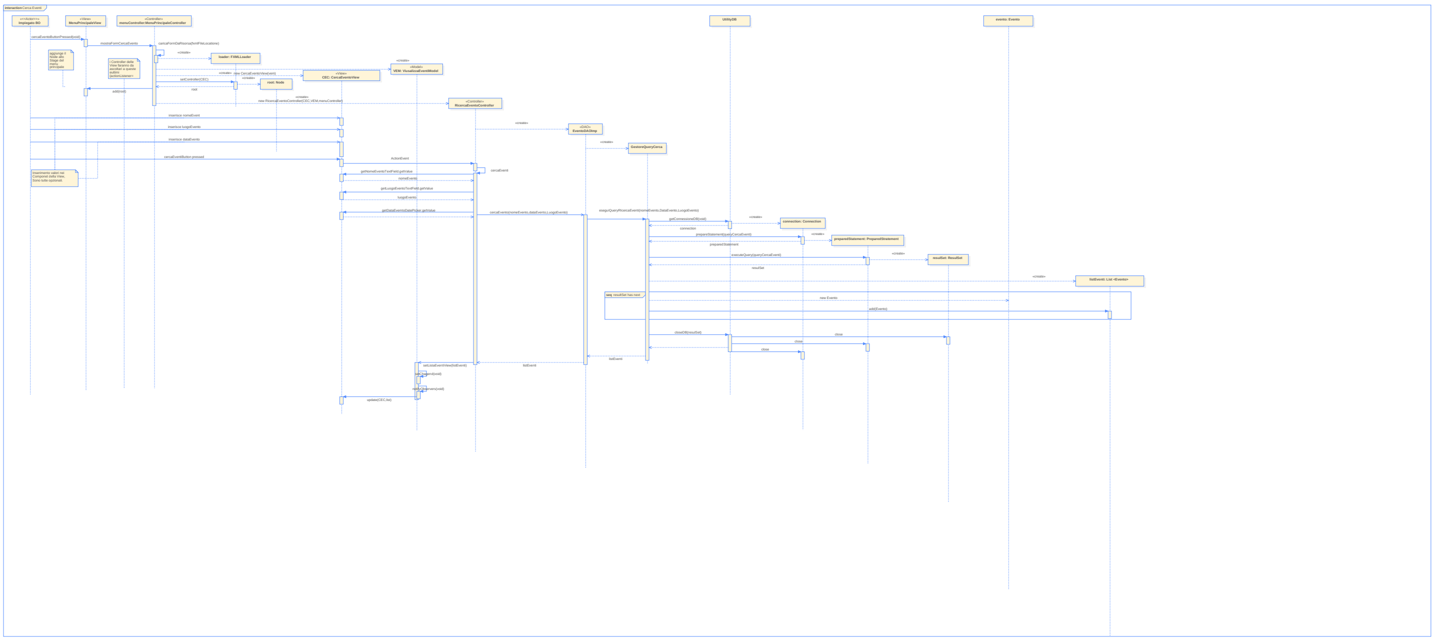
La Modifica di un evento avverrà nello stesso form, il quale ha già i campi necessari per la modifica, quando verrà scelto di modificare un evento, tutti i campi non utili ai fini della modifica verranno disabilitati e verrà attivati il tasto 'OK' il quale darà inizia alla procedura di modifica di un evento, il quale prevede la modifica di solo due campi : DataEvento e LuogoEvento.

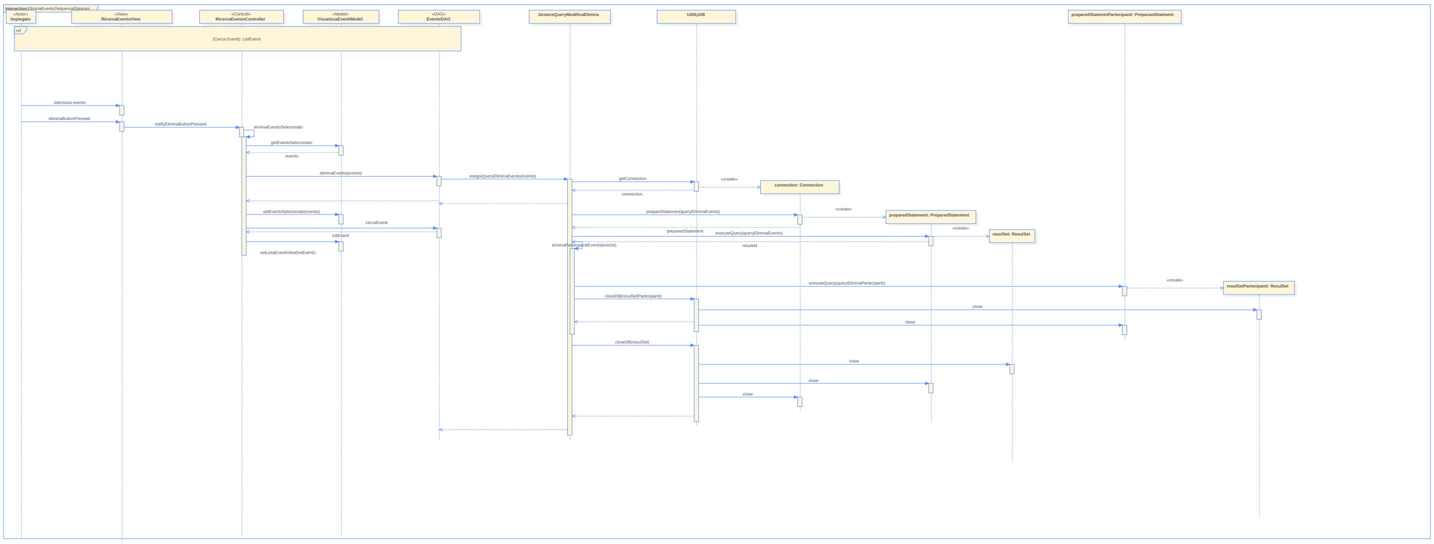
La Visualizzazione aprirà un nuovo form con tutti i dati di un evento, compreso l'insieme dei suoi partecipanti.

L'eliminazione avverrà solo se non saranno stati venduti biglietti di quel evento.

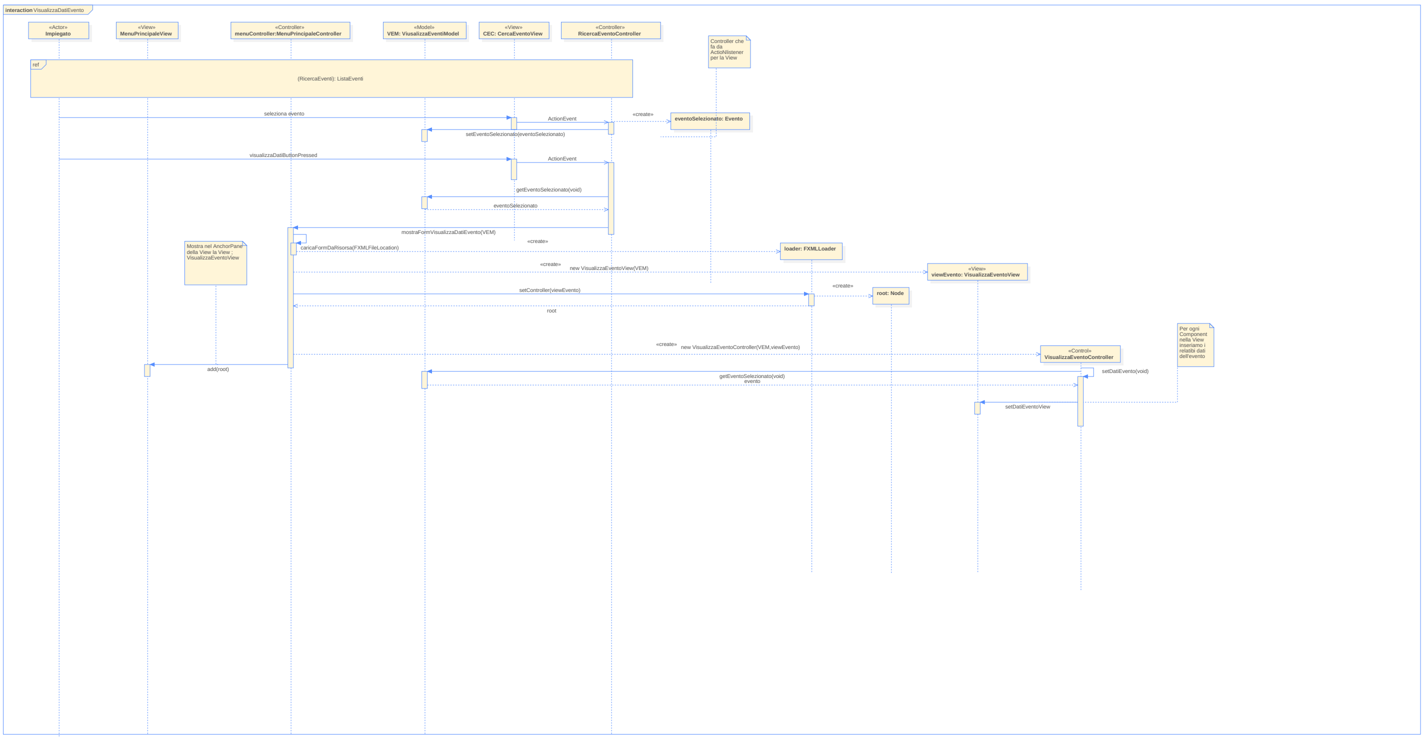
Mostriamo i **Class Diagram** e i **Sequence Diagram UML 2.0:**

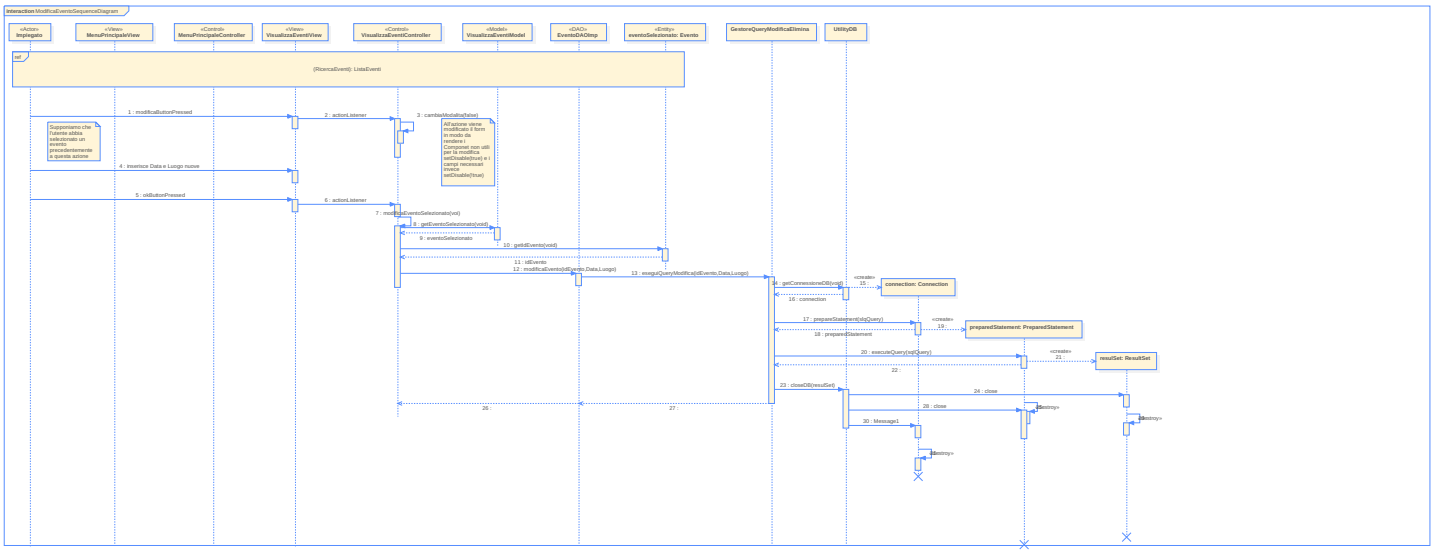

«View»
CercaEventoView
-cercaButton: Button
-nomeEventoTextField: TextField
-dataEventoTextField: TextField
-luogoEventoTextField: TextField
-tipologiaEvengoComboBox: ComboBox
-ricercaEventoController: RicercaEventoController
-modificaButton: Button
-eliminaButton: Button
-okButton: Button
-eventiModel: VisualizzaEventiModel
-tabellaCercaEventoTableView: TableView
-visualizzaDatiEventoButton: Button
+update(Observer model, Object lista): void
Observer
«View»
MenuPrincipaleView
(from Model)
-inserisciEventoButton: Button
-ricercaEventoButton: Button
-cambiaFormMenuPrincipale: AnchorPane
-gestioneClientiButton: Button
-visualizzaStatitisticheButton: Button
-gestioneDipidentiButton: Button
-paneView: AnchorPane
«View»
VisualizzaEventoView
-nomeEventoTextField: TextField
-luogoEventoTextField: TextField
-tipoEventoTextField: TextField
-genereEventoTextField: TextField
-okButton: Button
-prezzoTextField: TextField
-listaPartecipantiListView: ListView
+okButtonPressed(): void
Observer
Observable
«Model»
VisualizzaEventiModel
-listEventi: List
-eventoSelezionato: Evento
-setEventiListView(List)
-getEventiListView(): List
-setEventoSelezionato(Evento e): void
-getEventoSelezionato(void): Evento
ActionListener
«Controller»
MenuPrincipaleController
(from Model1)
-formCorrente: AnchorPane
-risorseForm: Map
-menuPrincipaleView: MenuPrincipaleView
-pulisciForm(): void
+mostraFormInserisciEvento(): void
-caricaRisorseForm(): void
+mostraFormGestioneClienti(): void
+mostraFormGestioneDipedenti(): void
+mostraFormVisualizzaStatistiche(): void
+mostraFormCercaEvento(): void
+mostraFormStatistiche(): Void
«Controller»
RicercaEventoController
-visualizzaEventiModel: VisualizzaEventiModel
-eventoDAO: EventoDAO
-cercaEventoView: CercaEventoView
-menuPrincipaleView: MenuPrincipaleView
+eliminaEventoSelezionato(Evento eventoSelezionato): void
+modificaEventoSelezionato(): void
+cercaEventi(String nomeEvento, LocalDate dataEvento, LuogoEnum luogoEvento): void
+modificaEventoSelezionato(Evento eventoSelezionato, LocalDate data, LuogoEnum luogo): void
-setListenerCercaView(): void
«Controller»
VisualizzaEventoController
-model: VisualizzEventoModel
-view: VisualizzaEventoView
-controller: MenuPrincipaleController
-setDatiEvento()
EventoDAO
(from Model1)
EventoDAOImpl
(from Model1)
-gestoreQueryInserimentoEvento: GestoreQueryInserimentoEvento
-gestoreQueryCerca: GestoreQueryCerca
-gestoreQueryModificaElimina: GestoreQueryModificaElimina
+inserisciEvento(Evento evento): void
+cercaEvento(LocaleDate date, LuogoEnum luogo, String nomeEvento): List
+eliminaEvento(Evento evento): void
+cercaEvento(): List
+modificaEvento(int eventoID, LocalDate data, LuogoEnum luogo): void
+cercaEventiPerCitta(String artista): map
+cercaEvento(String artista): List
GestoreQueryModificaElimina
+eseguiQueryModificaEvento(int eventoID, LocalDate data, LuogoEnum luogo): void
+eseguiQueryEliminaEvento(Evento eventoDaEliminare): void
-eseguiQueryEliminaPartecipantiEvento(Evento eventoDaEliminare): void
GestoreQueryCerca
+eseguiQueryRicercaClientii(String username): List
+eseguiQueryRicercaImpiegato(String nomeEvento, LuogoEnum luogoEvento, LocalDate dataEvento): List
+eseguiQueryRicercaEventi(): List
+eseguiQueryRicercaEventi(String nomeEvento, LuogoEnum luogoEvento, LocalDate dataEvento): List
-eseguiQueryRicercaPartecipantiEvento(int id): Set
+eseguiQueryRicercaImpiegatoConnesso(String username, String password): Impiegato
+eseguiQueryRicercaEventiArtista(String artista): List
+eseguiQueryRicercaEventiCitta(String artista): Map
UtilityDB
+getConnection(): Connection
+closeDB(PreparedStatement ps): void
DB
+crea
1
+ascolta
+aggiorna
+osserva
+usa
+modifica
+inserisce dati


interaction Cerca Eventi
«<<Actor>>» Impiegato BO
«View» MenuPrincipaleView
«Controller» menuController:MenuPrincipaleController
UtilityDB
evento: Evento
cercaEventoButtonPressed(void)
mostraFormCercaEvento
caricaFormDaRisorsa(fxmlFileLocation)
«create»
loader: FXMLLoader
aggiunge il Node allo Stage del menu principale
I Controller delle View faranno da ascoltari a queste eultimi (actionListener=
«create»
«Model» VEM: ViusalizzaEventiModel
«create» new CercaEventoView(vem)
«View» CEC: CercaEventoView
setController(CEC)
«create»
root: Node
root
add(root)
«create» new RicercaEventoController(CEC,VEM,menuController)
«Controller» RicercaEventoController
inserisce nomeEvent
inserisce luogoEvento
inserisce dataEvento
cercaEventiButton pressed
Inserimento valori nei Componet della View, Sono tutte opzionali.
«create»
«DAO» EventoDAOImp
«create»
GestoreQueryCerca
ActionEvent
cercaEventi
getNomeEventoTextField.getValue
nomeEvento
getLuogoEventoTextField.getValue
luogoEvento
getDataEventoDatePicker.getValue
cercaEvento(nomeEvento,dataEvento,LuogoEvento)
eseguiQueryRicercaEventi(nomeEvento,DataEvento,LuogoEvento)
getConnessioneDB(void)
«create»
connection: Connection
connection
prepareStatement(queryCercaEventi)
«create»
preparedStatement: PreparedStratement
preparedStatement
executeQuery(queryCercaEventi)
«create»
resulSet: ResulSet
resulSet
«create»
listEventi: List <Evento>
seq resultSet has next
new Evento
add(Evento)
closeDB(resulSet)
close
close
close
listEventi
listEventi
setListaEventView(listEventi)
setChagend(void)
notifyObservers(void)
update(CEC,list)



interaction EliminaEventoSequenceDiagram
«Actor» Impiegato
«View» RicercaEventoView
«Controll» RicercaEventoController
«Model» VisualizzaEventiModel
«DAO» EventoDAO
GestoreQueryModificaElimina
UtilityDB
praparedStatemtnPartecipanti: PreparaedStatment
ref
(Cerca Eventi): ListEventi
seleziona evento
eliminaButtonPressed
notifyEliminaButtonPressed
eliminaEventoSelezionato
getEventoSelezionato
evento
eliminaEvento(evento)
eseguiQueryEliminaEvento(evento)
getConnection
«create»
connection: Connection
connection
setEventoSelezionato(evento)
prepareStatemen(queryEliminaEvento)
«create»
preparedStatement: PreparedStatement
cercaEventi
preparedStatement
executeQuery(queryEliminaEvento)
«create»
resulSet: ResulSet
listEventi
eliminaPartecipantiEvento(evento)
resulset
setListaEventiView(listEventi)
exevuteQuery(queryEliminaPartecipanti)
«create»
resulSetPartecipanti: ResulSet
closeDB(resulSetPartecipanti)
close
close
closeDB(resulSet)
close
close
close


interaction VisualizzaDatiEvento
«Actor» Impiegato
«View» MenuPrincipaleView
«Controller» menuController:MenuPrincipaleController
«Model» VEM: ViusalizzaEventiModel
«View» CEC: CercaEventoView
«Controller» RicercaEventoController
Controller che fa da ActioNlistener per la View
ref
(RicercaEventi): ListaEventi
seleziona evento
ActionEvent
«create»
eventoSelezionato: Evento
setEventoSelezionato(eventoSelezionato)
visualizzaDatiButtonPressed
ActionEvent
getEventoSelezionato(void)
eventoSelezionato
mostraFormVisualizzaDatiEvento(VEM)
«create»
caricaFormDaRisorsa(FXMLFileLocation)
loader: FXMLLoader
Mostra nel AnchorPane della View la View ; VisualizzaEventoView
«create»
new VisualizzaEventoView(VEM)
«View» viewEvento: VisualizzaEventoView
setController(viewEvento)
«create»
root: Node
root
Per ogni Component nella View inseriamo i relatibi dati dell'evento
«create»
new VisualizzaEventoController(VEM,viewEvento)
«Control» VisualizzaEventoController
add(root)
setDatiEvento(void)
getEventoSelezionato(void)
evento
setDatiEventoView


interaction ModificaEventoSequenceDiagram
«Actor»
Impiegato
«View»
MenuPrincipaleView
«Control»
MenuPrincipaleController
«View»
VisualizzaEventiView
«Control»
VisualizzaEventiController
«Model»
VisualizzaEventiModel
«DAO»
EventoDAOimp
«Entity»
eventoSelezionato: Evento
GestoreQueryModificaElimina
UtilityDB
ref
(RicercaEventi): ListaEventi
1 : modificaButtonPressed
2 : actionListener
3 : cambiaModalita(false)
Supponiamo che l'utente abbia selezionato un evento precedentemente e questa azione
All'azione viene richiamato il form in modo di rendere i Componet non utili per la modifica setDisable(true) e i campi necessari setDisable(true)
4 : inserisce Data e Luogo nuove
5 : okButtonPressed
6 : actionListener
7 : modificaEventoSelezionato(void)
8 : getEventoSelezionato(void)
9 : eventoSelezionato
10 : getIdEvento(void)
11 : idEvento
12 : modificaEvento(idEvento,Data,Luogo)
13 : eseguiQueryModifica(idEvento,Data,Luogo)
14 : getConnessioneDB(void)
«create»
15 :
connection: Connection
16 : connection
17 : prepareStatement(sqlQuery)
«create»
18 :
preparedStatement: PreparedStatement
18 : preparedStatement
20 : executeQuery(sqlQuery)
«create»
21 :
resultSet: ResultSet
22 :
23 : closeDB(resultSet)
24 : close
«destroy»
25 :
26 : close
«destroy»
27 :
30 : Message1
«destroy»

Ricerca Evento CRC Card:

Elencate tutte le CRC card delle classi necessarie per la ricerca di un Evento e di conseguenza le altre funzionalità

Nome Classe: RicercaEventoController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener ai componenti della view CercaEvento, crea un costruttore ed impone funzioni di aggiornamento per alcuni componenti della ricerca ed infine i Metodi comunicanti con il DAO	**Collaboratori** VisualizzaEventiModel CercaEventoView EventoDAO MenuPrincipaleController

Nome Classe: VisualizzaEventoController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener e setter ai componenti della view	**Collaboratori** VisualizzaEventiModel VisualizzaEventoView MenuPrincipaleController

Nome Classe: VisualizzaEventiModel	**Package**: Model
Superclasses	Observable
Subclasses	-
Responsabilità Creare un setlist per gli eventi e un oggetto per l' eventi selezionato	**Collaboratori**

Nome Classe: CercaEventoView	**Package**: View
Superclasses	-
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata di ricerca di un evento gestendo i vari getter e di gestire il metodo update per la tabella di visualizzazione della lista	**Collaboratori** MenuPrincipaleController VisualizzaEventiModel RicercaEventoController

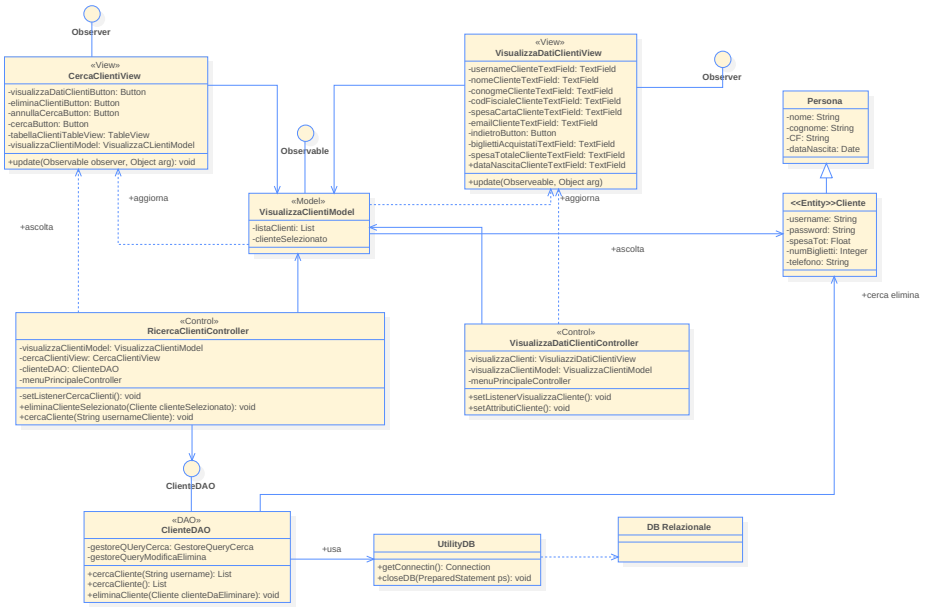
Nome Classe: VisualizzaEventoView	**Package**: View
Superclasses	Observer

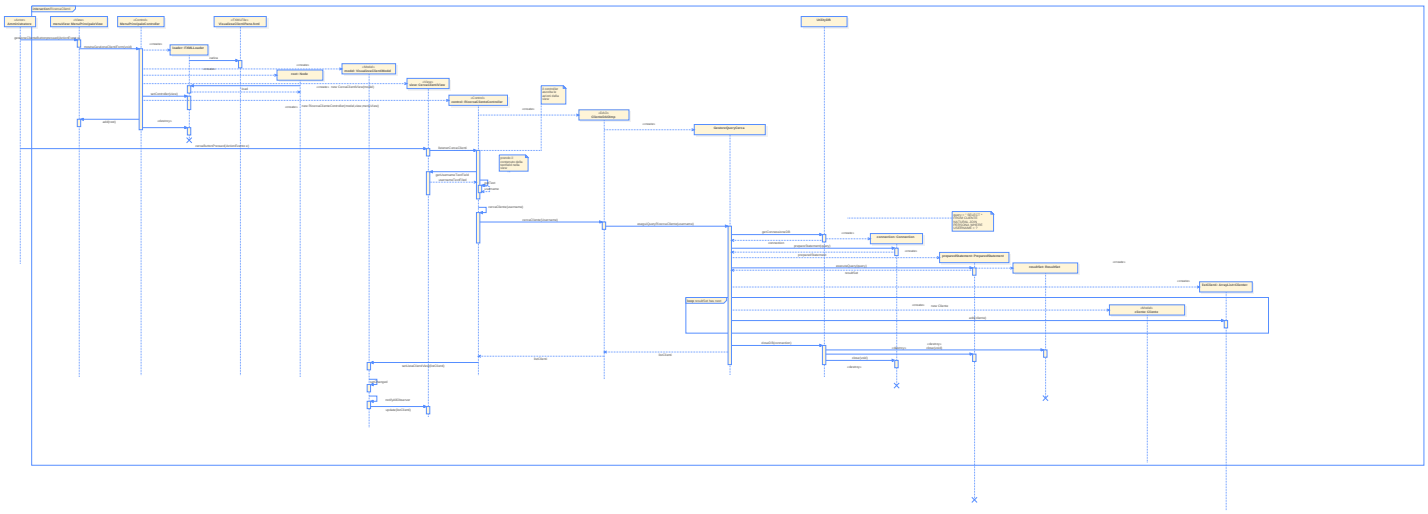
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata per visualizzare un evento gestendo i vari getter	**Collaboratori** VisualizzaEventiModel

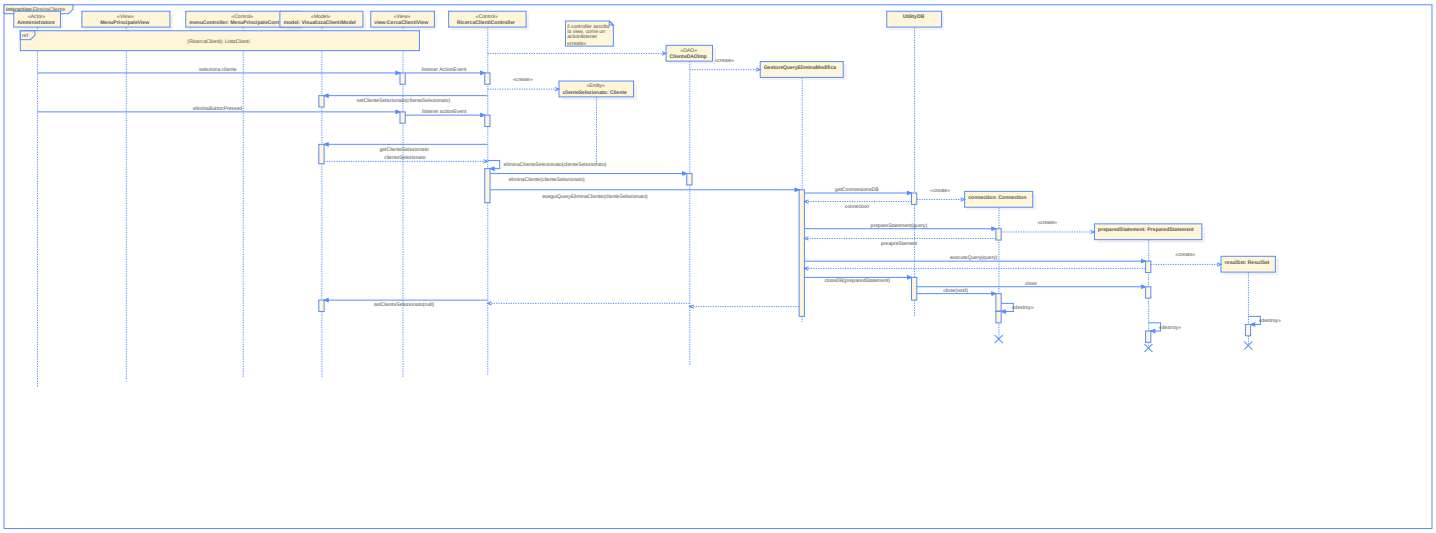
Ricerca Cliente:

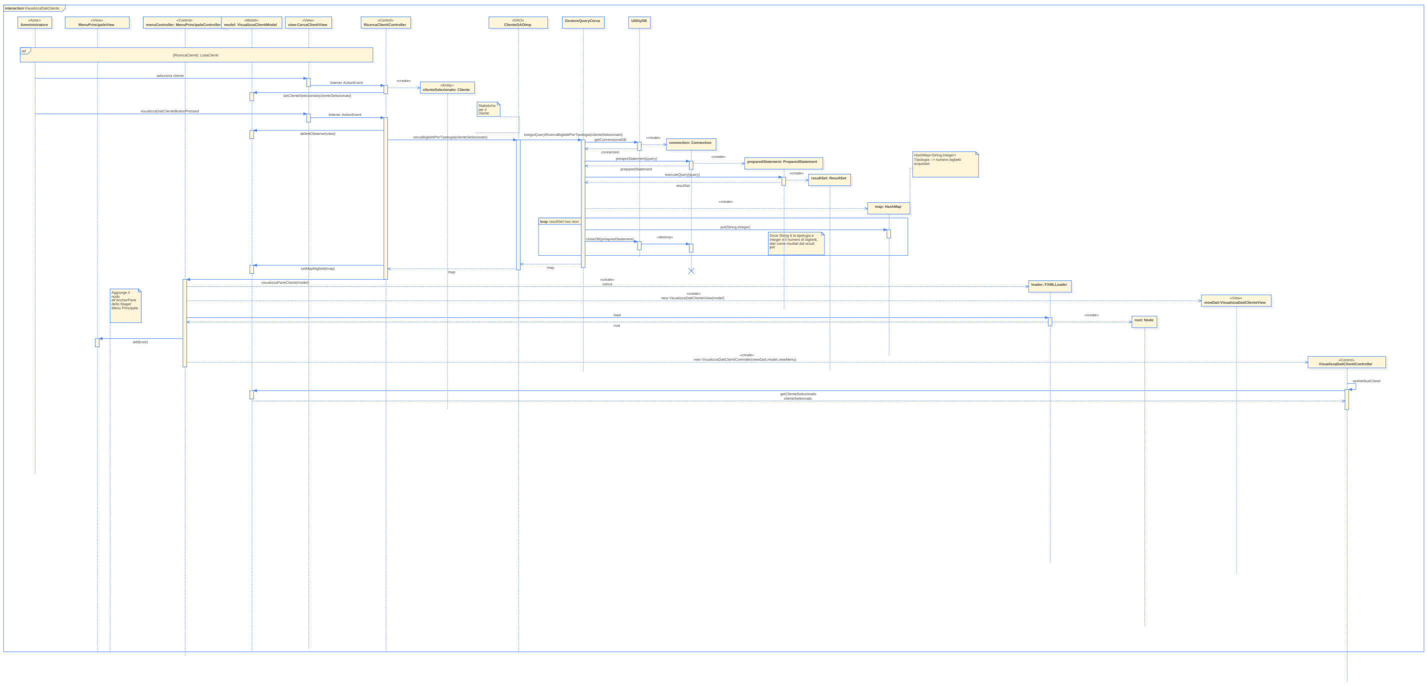
Per la ricerca di un cliente/i è stata usato lo stesso pattern e implementazioni simili a alla Ricerca di un Evento, la differenza qui sta nell'entità considerata, quindi il modello cambierà, e il numero di dati richiesti per la ricerca, abbiamo usato la ricerca per Username, verranno visualizzati i Clienti con l'Username esatto inserito oppure quelli che contengono la stringa inserita come sottostring del proprio username.
E' prevista solo l'eliminazione e la visualizzazione dei dati con relative statistiche del Cliente selezionato.

Diagrammi UML:


Observer
«View»
CercaClientiView
-visualizzaDatiClientiButton: Button
-eliminaClientiButton: Button
-annullaCercaButton: Button
-cercaButton: Button
-tabellaClientiTableView: TableView
-visualizzaClientiModel: VisualizzaCLientiModel
+update(Observable observer, Object arg): void
Observable
«View»
VisualizzaDatiClientiView
-usernameClienteTextField: TextField
-nomeClienteTextField: TextField
-conogmeClienteTextField: TextField
-codFiscaleClienteTextField: TextField
-spesaCartaClienteTextField: TextField
-emailClienteTextField: TextField
-indietroButton: Button
-bigliettiAcquistatiTextField: TextField
-spesaTotaleClienteTextField: TextField
+dataNascitaClienteTextField: TextField
+update(Observeable, Object arg)
Observer
Persona
-nome: String
-cognome: String
-CF: String
-dataNascita: Date
+aggiorna
+ascolta
«Model»
VisualizzaClientiModel
-listaClienti: List
-clienteSelezionato
+aggiorna
+ascolta
<<Entity>>Cliente
-username: String
-password: String
-spesaTot: Float
-numBiglietti: Integer
-telefono: String
+cerca elimina
«Control»
RicercaClientiController
-visualizzaClientiModel: VisualizzaClientiModel
-cercaClientiView: CercaClientiView
-clienteDAO: ClienteDAO
-menuPrincipaleController
-setListenerCercaClienti(): void
+eliminaClienteSelezionato(Cliente clienteSelezionato): void
+cercaCliente(String usernameCliente): void
«Control»
VisualizzaDatiClientiController
-visualizzaClienti: VisuliazziDatiClientiView
-visualizzaClientiModel: VisualizzaClientiModel
-menuPrincipaleController
+setListenerVisualizzaCliente(): void
+setAttributiCliente(): void
ClienteDAO
«DAO»
ClienteDAO
-gestoreQUeryCerca: GestoreQueryCerca
-gestoreQueryModificaElimina
+cercaCliente(String username): List
+cercaCliente(): List
+eliminaCliente(Cliente clienteDaEliminare): void
+usa
UtilityDB
+getConnectin(): Connection
+closeDB(PreparedStatement ps): void
DB Relazionale




Interaction EliminaCliente
«Actor»
Amministratore
«View»
MenuPrincipaleView
«Control»
menuController: MenuPrincipaleCont
«Model»
model: VisualizzaClientiModel
«View»
view:CercaClientiView
«Control»
RicercaClienteController
Il controller associato alla view, come un actionListener «create»
UtilityDB
alt
[RicercaClienti]: ListaClienti
«DAO»
ClienteDAOImp
«create»
GestoreQueryEliminaModificA
seleziona cliente
listener ActionEvent
«create»
«Entity»
clienteSelezionato: Cliente
setClienteSelezionato(clienteSelezionato)
eliminaButtonPressed
listener actionEvent
getClienteSelezionato
clienteSelezionato
eliminaClienteSelezionato(clienteSelezionato)
eliminaCliente(clienteSelezionato)
eseguiQueryEliminaCliente(clienteSelezionato)
getConnessioneDB
«create»
connection: Connection
connection
prepareStatement(query)
«create»
preparedStatement: PreparedStatement
preparedStatement
executeQuery(query)
«create»
resultSet: ResultSet
closeDB(preparedStatement)
close()
close
destroy»
destroy»
destroy»
setClienteSelezionato(null)


interaction VisualizzaDatiCliente
«Actor» Amministratore
«View» MenuPrincipaleView
«Control» menuController: MenuPrincipaleController
«Model» model: VisualizzaClientiModel
«View» view:CercaClientiView
«Control» RicercaClientiController
«DAO» ClienteDAOImp
GestoreQueryCerca
UtilityDB
ref
(RicercaClienti): ListaClienti
seleziona cliente
listener ActionEvent
«create»
«Entity» clienteSelezionato: Cliente
setClienteSelezionato(clienteSelezionato)
Statistiche per il cliente
visualizzaDatiClienteButtonPressed
listener ActionEvent
deleteObserver(view)
cercaBigliettiPerTipologia(clienteSelezionato)
eseguiQueryRicercaBigliettiPerTipologia(clienteSelezionato)
getConnessioneDB
«create»
connection: Connection
connection
preapreStatement(query)
«create»
preparedStatement: PreparedStatement
preparedStatement
executeQuery(query)
«create»
resultSet: ResultSet
resulSet
HashMap<String,Integer> Tipologia---> numero biglietti acquistati
«create»
map: HashMap
loop resultSet has next
put(String,Integer)
Dove String è la tipologia e integer è il numero di biglietti, dati come risultati dal result set:
closeDB(preapredStatement)
«destroy»
map
setMapNiglietti(map)
map
visualizzaPaneClienti(model)
«create»
carica
loader: FXMLLoader
«create»
new VisualizzaDatiClienteView(model)
«View» viewDati:VisualizzaDatiClienteView
Aggiunge il nodo all'AnchorPane dello Stager Menu Principale
load
root
«create»
root: Node
add(root)
«create»
new VisualizzaDatiClientiController(viewDati,model,viewMenu)
«Control» VisualizzaDatiClientiController
setAttributiClienti
getClienteSelezionato
clienteSelezionato

Gestione Cliente CRC Card

Nome Classe: Cliente	**Package**: Model
Superclasses	Persona
Subclasses	-
Responsabilità Genera un costruttore per creare un nuovo Cliennte e una stringa che associa ad ogni campo un attributo	**Collaboratori**

Nome Classe: CercaClientiview	**Package**: View
Superclasses	-
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata di ricerca per un cliente gestendo i vari getter e di gestire il metodo update per la tabella di visualizzazione della lista	**Collaboratori** MenuPrincipaleView RicercaClienteController VisualizzaClientiModel Cliente

Nome Classe: VisualizzaClientiModel	**Package**: Model
Superclasses	Observable
Subclasses	-
Responsabilità Creare un setlist per i clienti, un oggetto per il cliente selezionato e una mappa per i biglietti acquistati	**Collaboratori**

Nome Classe: VisualizzaDatiClientiView	**Package**: View
Superclasses	Observer
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata per visualizzare dei dati di un cliente gestendo i vari getter e di gestire il metodo update il grafico presente nella schermata	**Collaboratori** VisualizzaClientiModel

Nome Classe: RicercaClienteController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener ai componenti della view CercaClienti, impone delle operazione durante	**Collaboratori** VisualizzaClientiModel CercaClientiView

la chiamata di un metodo	ClientiDAO MenuPrincipaleController

Nome Classe: VisualizzaDatiClienteController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener e setter ai componenti della view E rende non editabile i componenti stessi	**Collaboratori** VisualizzaDatiClientiView VisualizzaClientiModel MenuPrincipaleController

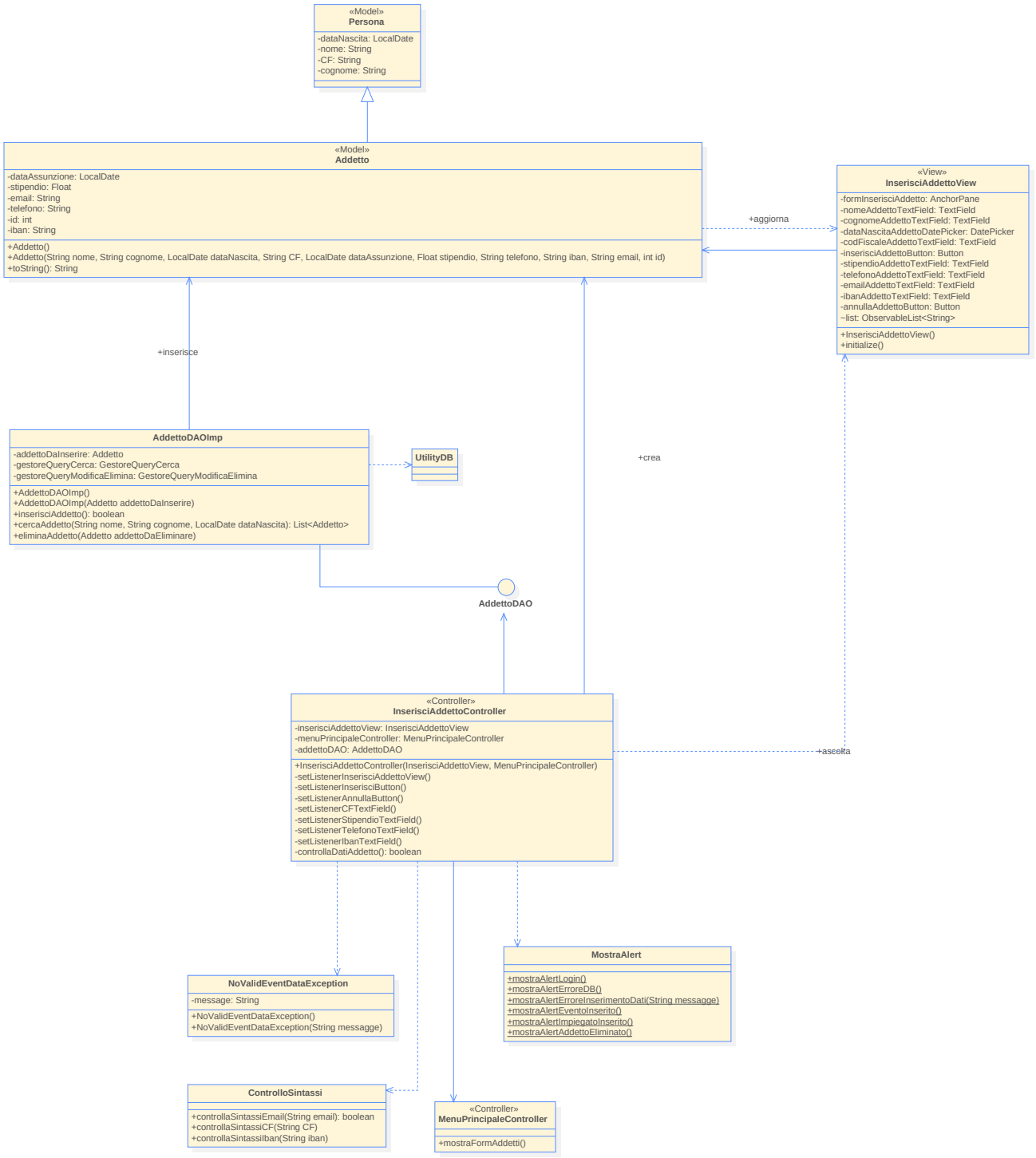
Nome Classe: ClientiDAOImp	**Package**:DB
Superclasses	ClientiDAO
Subclasses	-
Responsabilità Chiama le classi di GestioneQuery per eseguire le query relative alle Operazioni di C.R.U.D per l'entità *Cliente.*	**Collaboratori** gestoreQueryCerca gestoreQueryModificaElimina

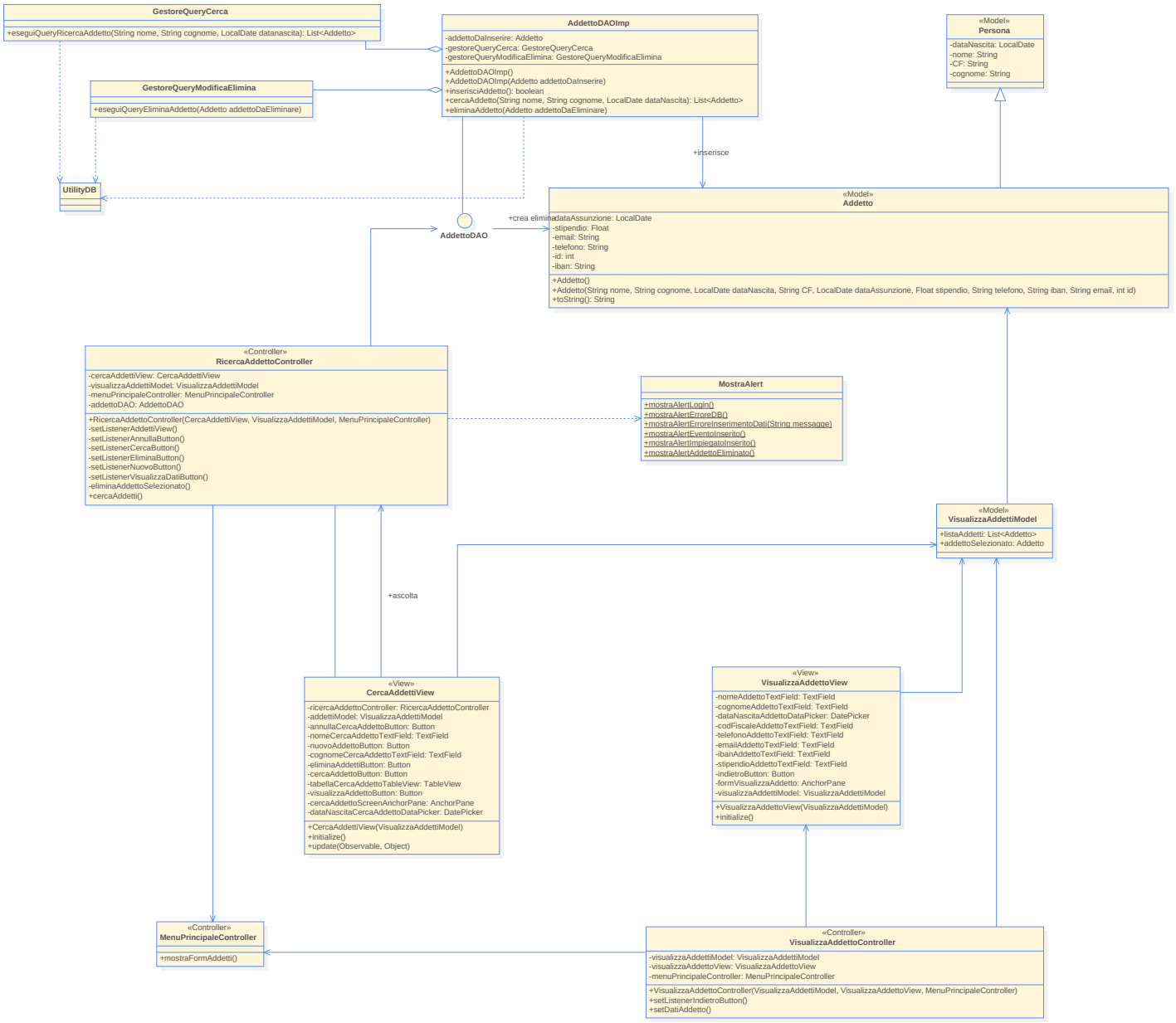
Gestione Addetti:

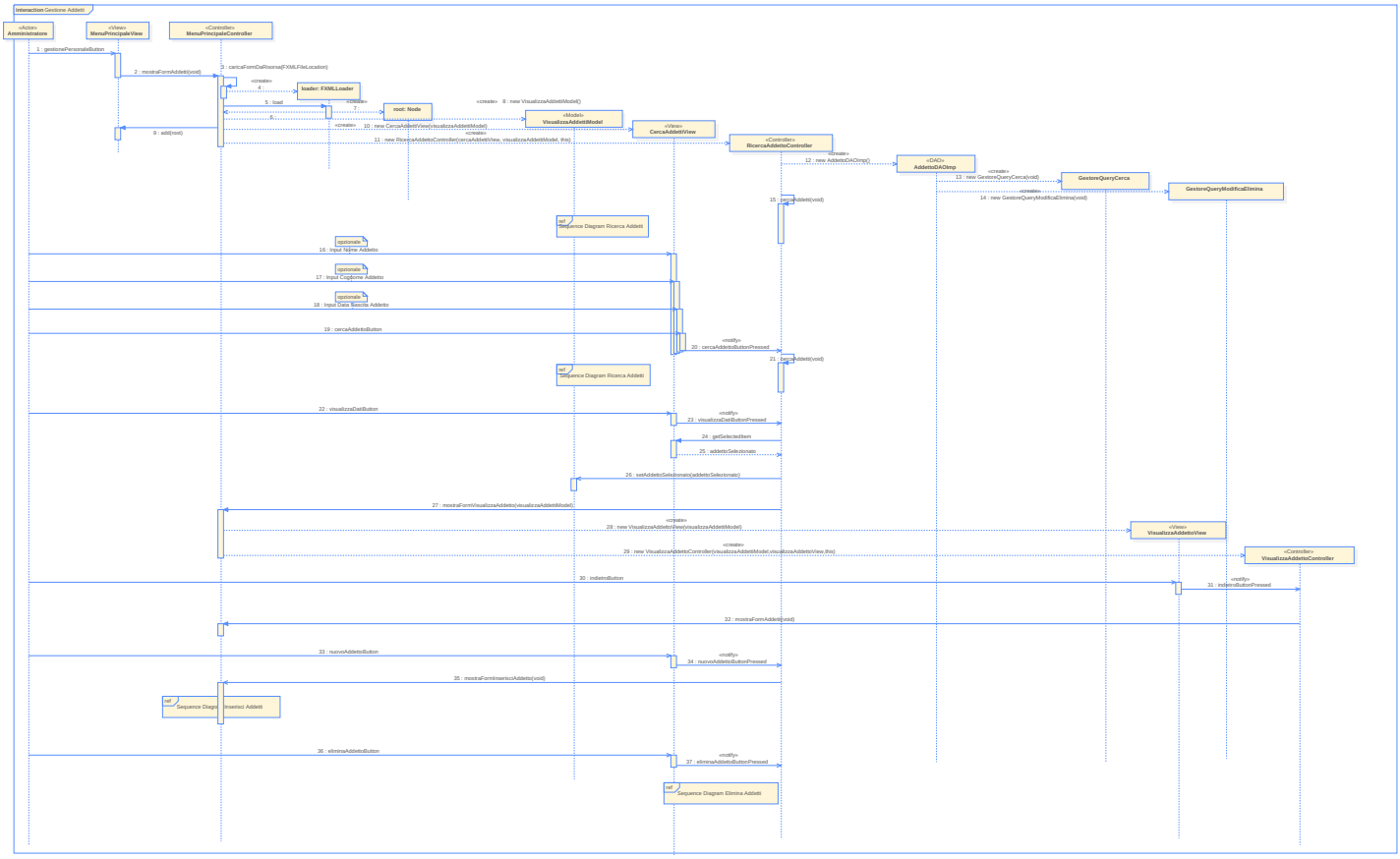
La Gestione Addetti è molto simile alla gestione dei Clienti, con la funzionalità aggiunta di inserimento di un addetto alla sicurezza, quindi useremo gli stessi pattern e le simili implementazioni cambiando di poco la logica.

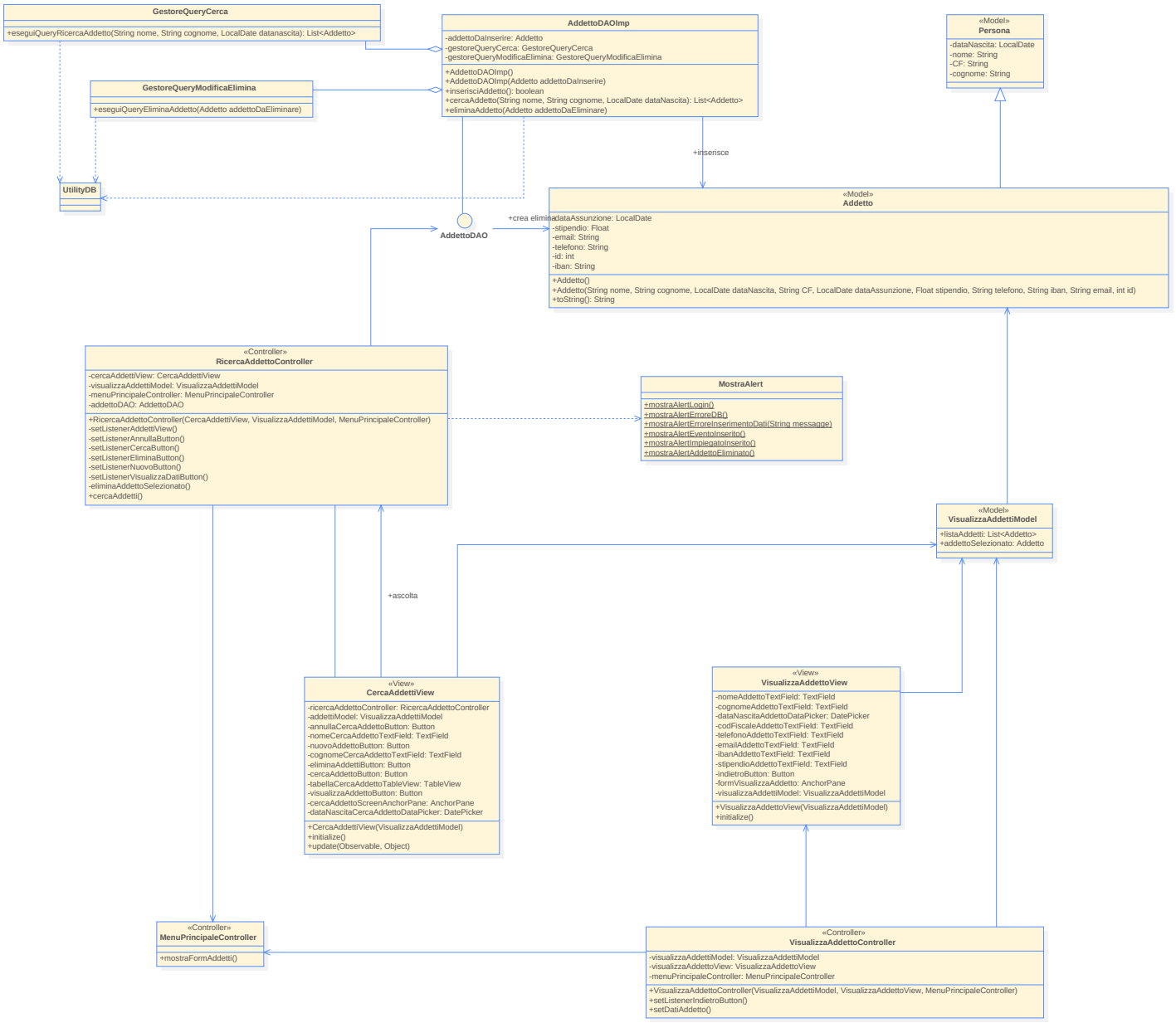
Questa macro funzionalità comprende la Ricerca di Addetti come per i Clienti, e oltre all inserimento già citato, è possibile eliminare un Addetto alla sicurezza dal Sistema.

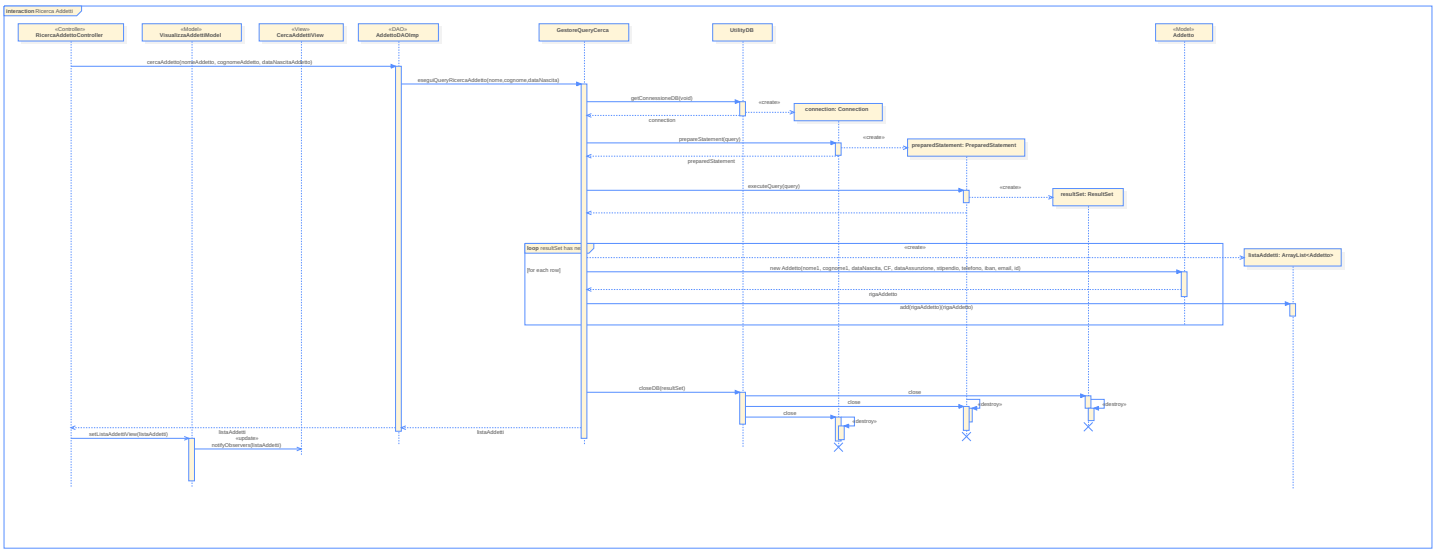
Diagrammi UML:

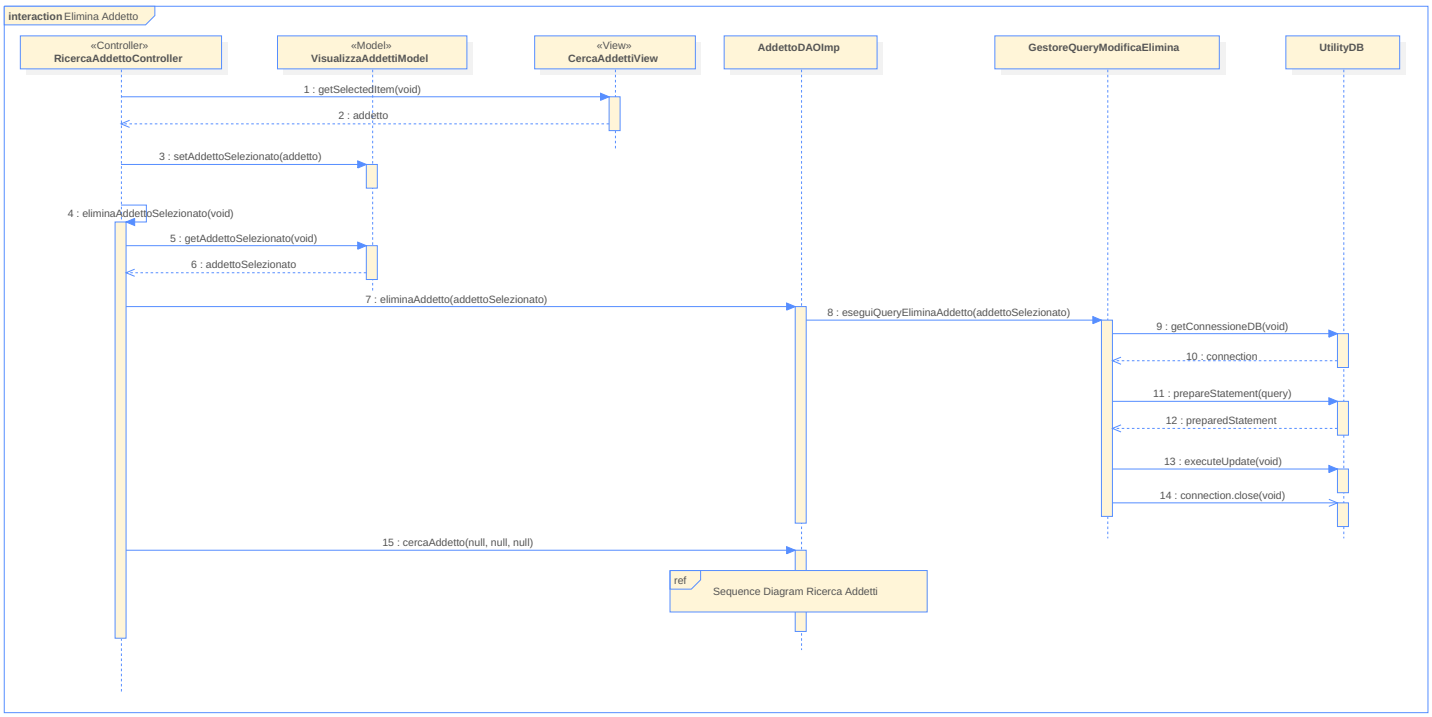

«Model»
Persona
-dataNascita: LocalDate
-nome: String
-CF: String
-cognome: String
«Model»
Addetto
-dataAssunzione: LocalDate
-stipendio: Float
-email: String
-telefono: String
-id: int
-iban: String
+Addetto()
+Addetto(String nome, String cognome, LocalDate dataNascita, String CF, LocalDate dataAssunzione, Float stipendio, String telefono, String iban, String email, int id)
+toString(): String
«View»
InserisciAddettoView
-formInserisciAddetto: AnchorPane
-nomeAddettoTextField: TextField
-cognomeAddettoTextField: TextField
-dataNascitaAddettoDatePicker: DatePicker
-codFiscaleAddettoTextField: TextField
-inserisciAddettoButton: Button
-stipendioAddettoTextField: TextField
-telefonoAddettoTextField: TextField
-emailAddettoTextField: TextField
-ibanAddettoTextField: TextField
-annullaAddettoButton: Button
~list: ObservableList<String>
+InserisciAddettoView()
+initialize()
+aggiorna
+inserisce
AddettoDAOImp
-addettoDaInserire: Addetto
-gestoreQueryCerca: GestoreQueryCerca
-gestoreQueryModificaElimina: GestoreQueryModificaElimina
+AddettoDAOImp()
+AddettoDAOImp(Addetto addettoDaInserire)
+inserisciAddetto(): boolean
+cercaAddetto(String nome, String cognome, LocalDate dataNascita): List<Addetto>
+eliminaAddetto(Addetto addettoDaEliminare)
UtilityDB
+crea
AddettoDAO
«Controller»
InserisciAddettoController
-inserisciAddettoView: InserisciAddettoView
-menuPrincipaleController: MenuPrincipaleController
-addettoDAO: AddettoDAO
+InserisciAddettoController(InserisciAddettoView, MenuPrincipaleController)
-setListenerInserisciAddettoView()
-setListenerInserisciButton()
-setListenerAnnullaButton()
-setListenerCFTextField()
-setListenerStipendioTextField()
-setListenerTelefonoTextField()
-setListenerIbanTextField()
-controllaDatiAddetto(): boolean
+ascolta
NoValidEventDataException
-message: String
+NoValidEventDataException()
+NoValidEventDataException(String messagge)
MostraAlert
+mostraAlertLogin()
+mostraAlertErroreDB()
+mostraAlertErroreInserimentoDati(String messagge)
+mostraAlertEventoInserito()
+mostraAlertImpiegatoInserito()
+mostraAlertAddettoEliminato()
ControlloSintassi
+controllaSintassiEmail(String email): boolean
+controllaSintassiCF(String CF)
+controllaSintassiIban(String iban)
«Controller»
MenuPrincipaleController
+mostraFormAddetti()

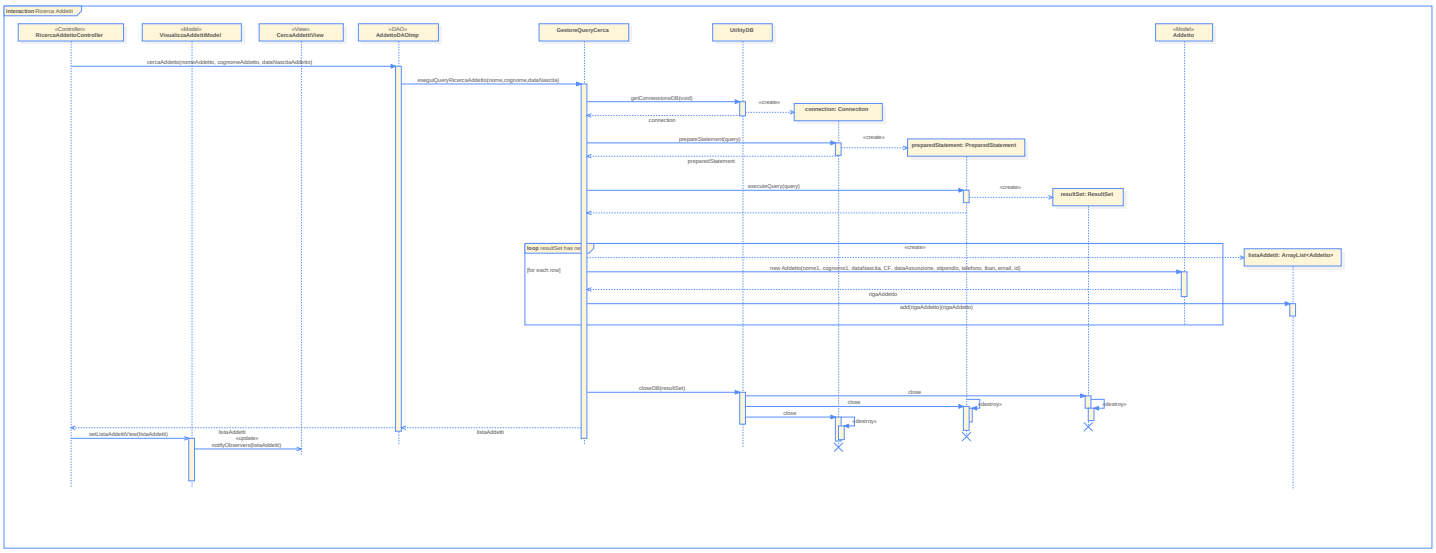

GestoreQueryCerca
+eseguiQueryRicercaAddetto(String nome, String cognome, LocalDate datanascita): List<Addetto>
AddettoDAOImp
-addettoDaInserire: Addetto
-gestoreQueryCerca: GestoreQueryCerca
-gestoreQueryModificaElimina: GestoreQueryModificaElimina
+AddettoDAOImp()
+AddettoDAOImp(Addetto addettoDaInserire)
+inserisciAddetto(): boolean
+cercaAddetto(String nome, String cognome, LocalDate dataNascita): List<Addetto>
+eliminaAddetto(Addetto addettoDaEliminare)
«Model»
Persona
-dataNascita: LocalDate
-nome: String
-CF: String
-cognome: String
GestoreQueryModificaElimina
+eseguiQueryEliminaAddetto(Addetto addettoDaEliminare)
+inserisce
UtilityDB
«Model»
Addetto
-dataAssunzione: LocalDate
-stipendio: Float
-email: String
-telefono: String
-id: int
-iban: String
+Addetto()
+Addetto(String nome, String cognome, LocalDate dataNascita, String CF, LocalDate dataAssunzione, Float stipendio, String telefono, String iban, String email, int id)
+toString(): String
+crea elimina
AddettoDAO
«Controller»
RicercaAddettoController
-cercaAddettiView: CercaAddettiView
-visualizzaAddettiModel: VisualizzaAddettiModel
-menuPrincipaleController: MenuPrincipaleController
-addettoDAO: AddettoDAO
+RicercaAddettoController(CercaAddettiView, VisualizzaAddettiModel, MenuPrincipaleController)
-setListenerAddettiView()
-setListenerAnnullaButton()
-setListenerCercaButton()
-setListenerEliminaButton()
-setListenerNuovoButton()
-setListenerVisualizzaDatiButton()
-eliminaAddettoSelezionato()
+cercaAddetti()
MostraAlert
+mostraAlertLogin()
+mostraAlertErroreDB()
+mostraAlertErroreInserimentoDati(String messagge)
+mostraAlertEventoInserito()
+mostraAlertImpiegatoInserito()
+mostraAlertAddettoEliminato()
«Model»
VisualizzaAddettiModel
+listaAddetti: List<Addetto>
+addettoSelezionato: Addetto
+ascolta
«View»
CercaAddettiView
-ricercaAddettoController: RicercaAddettoController
-addettiModel: VisualizzaAddettiModel
-annullaCercaAddettoButton: Button
-nomeCercaAddettoTextField: TextField
-nuovoAddettoButton: Button
-cognomeCercaAddettoTextField: TextField
-eliminaAddettiButton: Button
-cercaAddettoButton: Button
-tabellaCercaAddettoTableView: TableView
-visualizzaAddettoButton: Button
-cercaAddettoScreenAnchorPane: AnchorPane
-dataNascitaCercaAddettoDataPicker: DatePicker
+CercaAddettiView(VisualizzaAddettiModel)
+initialize()
+update(Observable, Object)
«View»
VisualizzaAddettoView
-nomeAddettoTextField: TextField
-cognomeAddettoTextField: TextField
-dataNascitaAddettoDataPicker: DatePicker
-codFiscaleAddettoTextField: TextField
-telefonoAddettoTextField: TextField
-emailAddettoTextField: TextField
-ibanAddettoTextField: TextField
-stipendioAddettoTextField: TextField
-indietroButton: Button
-formVisualizzaAddetto: AnchorPane
-visualizzaAddettiModel: VisualizzaAddettiModel
+VisualizzaAddettoView(VisualizzaAddettiModel)
+initialize()
«Controller»
MenuPrincipaleController
+mostraFormAddetti()
«Controller»
VisualizzaAddettoController
-visualizzaAddettiModel: VisualizzaAddettiModel
-visualizzaAddettoView: VisualizzaAddettoView
-menuPrincipaleController: MenuPrincipaleController
+VisualizzaAddettoController(VisualizzaAddettiModel, VisualizzaAddettoView, MenuPrincipaleController)
+setListenerIndietroButton()
+setDatiAddetto()


Interaction Gestione Addetti
«Actor» Amministratore
«View» MenuPrincipaleView
«Controller» MenuPrincipaleController
1 : gestionePersonaleButton
2 : mostraFormAddetti(void)
3 : caricaFormDaRisorsa(FXMLFileLocation)
«create» 4
loader: FXMLLoader
5 : load
«create»
root: Node
«create» 8 : new VisualizzaAddettiModel()
«Model» VisualizzaAddettiModel
«create» 10 : new CercaAddettiView(visualizzaAddettiModel)
«View» CercaAddettiView
9 : add(root)
11 : new RicercaAddettoController(cercaAddettiView, visualizzaAddettiModel, this)
«Controller» RicercaAddettoController
«create» 12 : new AddettoDAOimpl()
«DAO» AddettoDAOimp
«create» 13 : new GestoreQueryCerca(void)
«create» 14 : new GestoreQueryModificaElimina(void)
GestoreQueryCerca
GestoreQueryModificaElimina
15 : cercaAddetti(void)
ref Sequence Diagram Ricerca Addetti
opzionale 16 : Input Nome Addetto
opzionale 17 : Input Cognome Addetto
opzionale 18 : Input Data Nascita Addetto
19 : cercaAddettoButton
«notify» 20 : cercaAddettoButtonPressed
21 : cercaAddetti(void)
ref Sequence Diagram Ricerca Addetti
22 : visualizzaDatiButton
«notify» 23 : visualizzaDatiButtonPressed
24 : getSelectedItem
25 : addettoSelezionato
26 : setAddettoSelezionato(addettoSelezionato)
27 : mostraFormVisualizzaAddetto(visualizzaAddettiModel)
«create» 28 : new VisualizzaAddettoView(visualizzaAddettiModel)
«View» VisualizzaAddettoView
«create» 29 : new VisualizzaAddettoController(visualizzaAddettiModel,visualizzaAddettoView,this)
«Controller» VisualizzaAddettoController
30 : indietroButton
«notify» 31 : indietroButtonPressed
32 : mostraFormAddetti(void)
33 : nuovoAddettoButton
«notify» 34 : nuovoAddettoButtonPressed
35 : mostraFormInserisciAddetto(void)
ref Sequence Diagra
Inserisci Addetti
36 : eliminaAddettoButton
«notify» 37 : eliminaAddettoButtonPressed
ref Sequence Diagram Elimina Addetti


GestoreQueryCerca
+eseguiQueryRicercaAddetto(String nome, String cognome, LocalDate datanascita): List<Addetto>
AddettoDAOImp
-addettoDaInserire: Addetto
-gestoreQueryCerca: GestoreQueryCerca
-gestoreQueryModificaElimina: GestoreQueryModificaElimina
+AddettoDAOImp()
+AddettoDAOImp(Addetto addettoDaInserire)
+inserisciAddetto(): boolean
+cercaAddetto(String nome, String cognome, LocalDate dataNascita): List<Addetto>
+eliminaAddetto(Addetto addettoDaEliminare)
«Model»
Persona
-dataNascita: LocalDate
-nome: String
-CF: String
-cognome: String
GestoreQueryModificaElimina
+eseguiQueryEliminaAddetto(Addetto addettoDaEliminare)
+inserisce
UtilityDB
«Model»
Addetto
-dataAssunzione: LocalDate
-stipendio: Float
-email: String
-telefono: String
-id: int
-iban: String
+Addetto()
+Addetto(String nome, String cognome, LocalDate dataNascita, String CF, LocalDate dataAssunzione, Float stipendio, String telefono, String iban, String email, int id)
+toString(): String
+crea elimina
AddettoDAO
«Controller»
RicercaAddettoController
-cercaAddettiView: CercaAddettiView
-visualizzaAddettiModel: VisualizzaAddettiModel
-menuPrincipaleController: MenuPrincipaleController
-addettoDAO: AddettoDAO
+RicercaAddettoController(CercaAddettiView, VisualizzaAddettiModel, MenuPrincipaleController)
-setListenerAddettiView()
-setListenerAnnullaButton()
-setListenerCercaButton()
-setListenerEliminaButton()
-setListenerNuovoButton()
-setListenerVisualizzaDatiButton()
-eliminaAddettoSelezionato()
+cercaAddetti()
MostraAlert
+mostraAlertLogin()
+mostraAlertErroreDB()
+mostraAlertErroreInserimentoDati(String messagge)
+mostraAlertEventoInserito()
+mostraAlertImpiegatoInserito()
+mostraAlertAddettoEliminato()
«Model»
VisualizzaAddettiModel
+listaAddetti: List<Addetto>
+addettoSelezionato: Addetto
+ascolta
«View»
CercaAddettiView
-ricercaAddettoController: RicercaAddettoController
-addettiModel: VisualizzaAddettiModel
-annullaCercaAddettoButton: Button
-nomeCercaAddettoTextField: TextField
-nuovoAddettoButton: Button
-cognomeCercaAddettoTextField: TextField
-eliminaAddettiButton: Button
-cercaAddettoButton: Button
-tabellaCercaAddettoTableView: TableView
-visualizzaAddettoButton: Button
-cercaAddettoScreenAnchorPane: AnchorPane
-dataNascitaCercaAddettoDataPicker: DatePicker
+CercaAddettiView(VisualizzaAddettiModel)
+initialize()
+update(Observable, Object)
«View»
VisualizzaAddettoView
-nomeAddettoTextField: TextField
-cognomeAddettoTextField: TextField
-dataNascitaAddettoDataPicker: DatePicker
-codFiscaleAddettoTextField: TextField
-telefonoAddettoTextField: TextField
-emailAddettoTextField: TextField
-ibanAddettoTextField: TextField
-stipendioAddettoTextField: TextField
-indietroButton: Button
-formVisualizzaAddetto: AnchorPane
-visualizzaAddettiModel: VisualizzaAddettiModel
+VisualizzaAddettoView(VisualizzaAddettiModel)
+initialize()
«Controller»
MenuPrincipaleController
+mostraFormAddetti()
«Controller»
VisualizzaAddettoController
-visualizzaAddettiModel: VisualizzaAddettiModel
-visualizzaAddettoView: VisualizzaAddettoView
-menuPrincipaleController: MenuPrincipaleController
+VisualizzaAddettoController(VisualizzaAddettiModel, VisualizzaAddettoView, MenuPrincipaleController)
+setListenerIndietroButton()
+setDatiAddetto()


interaction Ricerca Addetti
«Controller»
RicercaAddettoController
«Model»
VisualizzaAddettiModel
«View»
CercaAddettiView
«DAO»
AddettoDAOimp
GestoreQueryCerca
UtilityDB
«Model»
Addetto
cercaAddetto(nomeAddetto, cognomeAddetto, dataNascitaAddetto)
eseguiQueryRicercaAddetto(nome,cognome,dataNascita)
getConnessioneDB(void)
«create»
connection: Connection
connection
prepareStatement(query)
«create»
preparedStatement: PreparedStatement
preparedStatement
executeQuery(query)
«create»
resultSet: ResultSet
loop resultSet has ne
«create»
listaAddetti: ArrayList<Addetto>
[for each row]
new Addetto(nome1, cognome1, dataNascita, CF, dataAssunzione, stipendio, telefono, iban, email, id)
rigaAddetto
add(rigaAddetto)(rigaAddetto)
closeDB(resultSet)
close
«destroy»
close
«destroy»
close
«destroy»
setListaAddettiView(listaAddetti)
listaAddetti
«update»
notifyObservers(listaAddetti)
listaAddetti


interaction Elimina Addetto
«Controller»
RicercaAddettoController
«Model»
VisualizzaAddettiModel
«View»
CercaAddettiView
AddettoDAOImp
GestoreQueryModificaElimina
UtilityDB
1 : getSelectedItem(void)
2 : addetto
3 : setAddettoSelezionato(addetto)
4 : eliminaAddettoSelezionato(void)
5 : getAddettoSelezionato(void)
6 : addettoSelezionato
7 : eliminaAddetto(addettoSelezionato)
8 : eseguiQueryEliminaAddetto(addettoSelezionato)
9 : getConnessioneDB(void)
10 : connection
11 : prepareStatement(query)
12 : preparedStatement
13 : executeUpdate(void)
14 : connection.close(void)
15 : cercaAddetto(null, null, null)
ref
Sequence Diagram Ricerca Addetti


interaction Ricerca Addetti
«Controller»
RicercaAddettoController
«Model»
VisualizzaAddettiModel
«View»
CercaAddettiView
«DAO»
AddettoDAOimp
GestoreQueryCerca
UtilityDB
«Model»
Addetto
cercaAddetto(nomeAddetto, cognomeAddetto, dataNascitaAddetto)
eseguiQueryRicercaAddetto(nome,cognome,dataNascita)
getConnessioneDB(void)
«create»
connection: Connection
connection
prepareStatement(query)
«create»
preparedStatement: PreparedStatement
preparedStatement
executeQuery(query)
«create»
resultSet: ResultSet
loop resultSet has ne
«create»
listaAddetti: ArrayList<Addetto>
[for each row]
new Addetto(nome1, cognome1, dataNascita, CF, dataAssunzione, stipendio, telefono, iban, email, id)
rigaAddetto
add(rigaAddetto)(rigaAddetto)
closeDB(resultSet)
close
«destroy»
close
«destroy»
close
«destroy»
setListaAddettiView(listaAddetti)
listaAddetti
«update»
notifyObservers(listaAddetti)
listaAddetti

Gestione Addetti CRC Card:

Nome Classe: AddettoDAOImp	**Package**:DB
Superclasses	AddettoDAO
Subclasses	-
Responsabilità Chiama le classi di GestioneQuery per eseguire le query relative alle Operazioni di C.R.U.D per l'entità *Addetto,* nella classe stessa cè anche una funzione per creare un nuovo addetto .	**Collaboratori** gestoreQueryCerca gestoreQueryModificaElimina

Nome Classe: Addetto	**Package**:Model
Superclasses	Persona
Subclasses	-
Responsabilità Genera un costruttore per creare un nuovo Addetto e una stringa che associa ad ogni campo un attributo	**Collaboratori**

Nome Classe: VisualizzaAddettiModel	**Package**: Model
Superclasses	Observable
Subclasses	-
Responsabilità Creare un setlist per gli addetti e un oggetto per l'addetto selezionato	**Collaboratori**

Nome Classe: VisualizzaAddettoController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener e setter ai componenti della view E rende non editabile i componenti stessi	**Collaboratori** VisualizzaAddettiModel VisualizzaAddettoView MenuPrincipaleController

Nome Classe: CercaAddettiView	**Package**: View
Superclasses	-
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata di ricerca per un addetto gestendo i vari getter e di gestire il metodo update per la tabella di visualizzazione della lista	**Collaboratori** RicercaAddettoController VisualizzaAddettiModel

Nome Classe: InserisciAddettoController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge Listener ed EventHandler ai componente della View, controllando la logica dell inserimento di un Addetto, controllando che i vincoli di un evento siano stati rispettati, creare un Evento.	**Collaboratori:** MenuPrincipaleController AddettoDAO InserisciAddettoView

Nome Classe: RicercaAddettoController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener ai componenti della view CercaAddetti, crea un costruttore ed impone funzioni di aggiornamento per alcuni componenti della ricerca ed infine i Metodi comunicanti con il DAO	**Collaboratori** CercaAddettiView VisualizzaAddettiModel MenuPrincipaleController AddettoDAO

Nome Classe: VisualizzaAddettoView	**Package**: View
Superclasses	-
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata per visualizzare gli Addetti gestendo i vari getter	**Collaboratori** VisualizzaAddettiModel

Nome Classe: InserisciAddettiView	**Package**: View
Superclasses	-
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata di inserimento di un addetto	**Collaboratori**

Statistiche:
Per le statistiche è stato utilizzato come Design Pattern **MVC** e Observer.

Abbiamo descritto le seguenti statistiche:

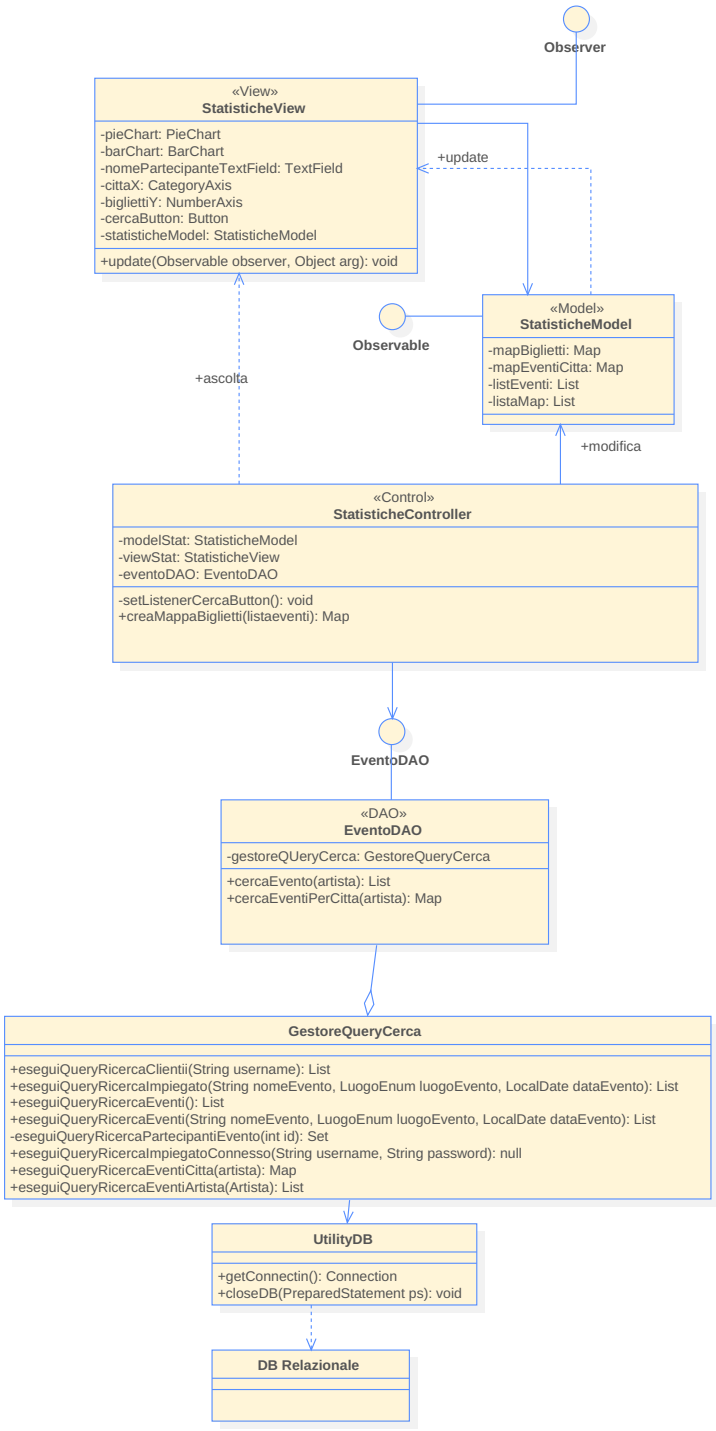
- *Numero Biglietti Venduti per Città di un Artista/Team*
- *Numero di Concerti per Città di un Artista*
- *Numero di biglietti acquistati da un Client divisi per Tipologia*

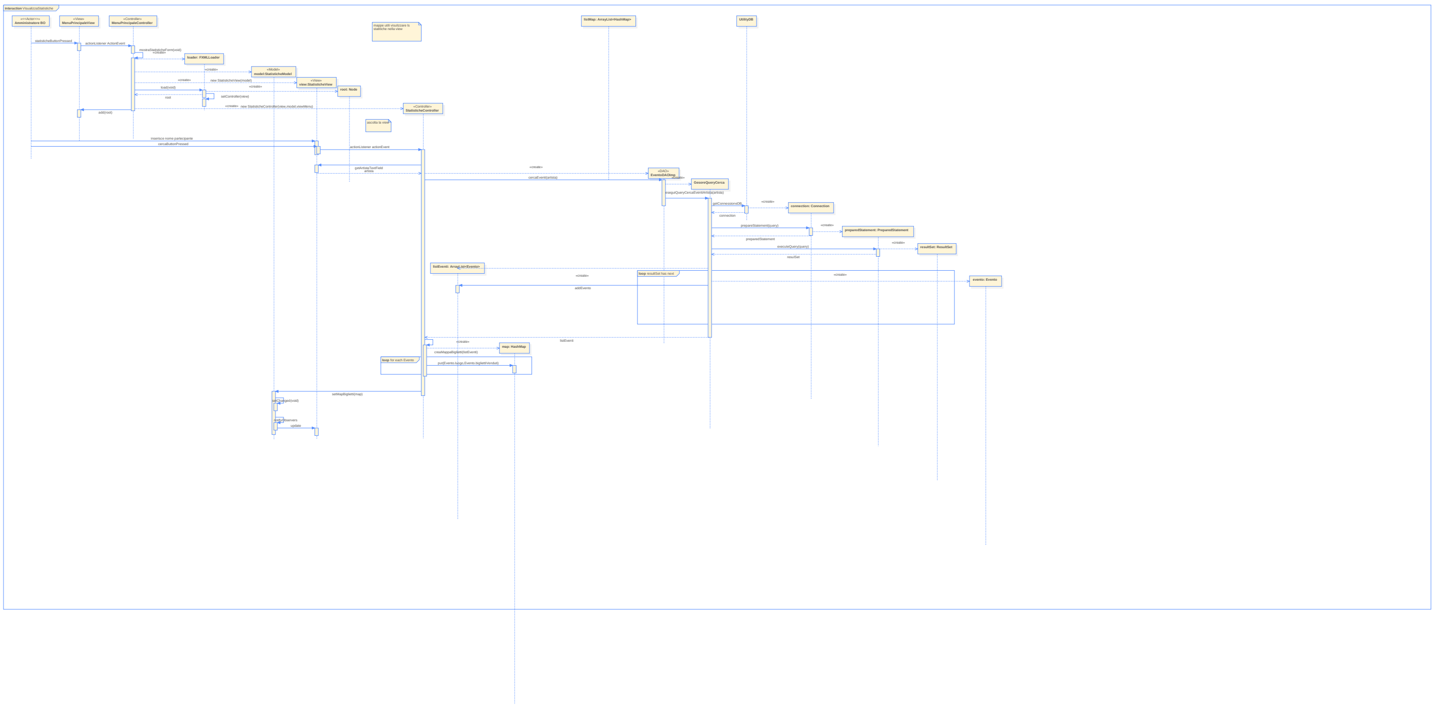
Le prime due statistiche vengono mostrate nella schermata apposita a cui si puà arrivare dal menu principale, la l'ultima, verrà visualizzata quando si vorrà visualizzare un Cliente dopo una ricerca, e nella schermata apposita di visualizzazione dei dati verrà caricato un PieChart con le sue statistiche.

L'utente dovrà inserire una stringa nel campo di ricerca e il sistema farà una ricerca nel DB per trovare eventi con quel partecipante inserito.

Per la prima statistica è possibile confrontare più Artisti/Team/Partecipanti, il numero di confronti è illimitato.

Mostriamo ora i **Diagrammi UML**:


Observer
«View»
Statistiche View
-pieChart: PieChart
-barChart: BarChart
-nomePartecipanteTextField: TextField
-cittaX: CategoryAxis
-biglietti Y: NumberAxis
-cercaButton: Button
-statisticheModel: StatisticheModel
+update(Observable observer, Object arg): void
+update
Observable
«Model»
StatisticheModel
-mapBiglietti: Map
-mapEventiCitta: Map
-listEventi: List
-listaMap: List
+ascolta
+modifica
«Control»
StatisticheController
-modelStat: StatisticheModel
-viewStat: StatisticheView
-eventoDAO: EventoDAO
-setListenerCercaButton(): void
+creaMappaBiglietti(listaeventi): Map
EventoDAO
«DAO»
EventoDAO
-gestoreQUeryCerca: GestoreQueryCerca
+cercaEvento(artista): List
+cercaEventiPerCitta(artista): Map
GestoreQueryCerca
+eseguiQueryRicercaClientii(String username): List
+eseguiQueryRicercaImpiegato(String nomeEvento, LuogoEnum luogoEvento, LocalDate dataEvento): List
+eseguiQueryRicercaEventi(): List
+eseguiQueryRicercaEventi(String nomeEvento, LuogoEnum luogoEvento, LocalDate dataEvento): List
-eseguiQueryRicercaPartecipantiEvento(int id): Set
+eseguiQueryRicercaImpiegatoConnesso(String username, String password): null
+eseguiQueryRicercaEventiCitta(artista): Map
+eseguiQueryRicercaEventiArtista(Artista): List
UtilityDB
+getConnectin(): Connection
+closeDB(PreparedStatement ps): void
DB Relazionale


interaction VisualizzaStatistiche
«<<Actor>>»
Amministratore BO
«View»
MenuPrincipaleView
«Controller»
MenuPrincipaleController
mappe utili visulizzare le statitiche nella view
listMap: ArrayList<HashMap>
UtilityDB
statisticheButtonPressed
actionListener ActionEvent
mostraStatisticheForm(void)
«create»
loader: FXMLLoader
«create»
«Model»
model:StatisticheModel
«create»
new StatisticheView(model)
«View»
view:StatisticheView
load(void)
«create»
root: Node
root
setController(view)
«create»
new StatisticheController(view,model,viewMenu)
«Controller»
StatisticheController
add(root)
ascolta la view
inserisce nome partecipante
cercaButtonPressed
actionListener actionEvent
getArtistaTextField
artista
«create»
«DAO»
EventoDAOImp
cercaEventi(artista)
«create»
GesoreQueryCerca
eseguiQueryCercaEventiArtista(artista)
getConnessioneDB
«create»
connection: Connection
connection
prepareStatement(query)
«create»
preparedStatement: PreparedStatement
preparedStatement
executeQuery(query)
«create»
resultSet: ResultSet
resultSet
listEventi: ArrayList<Evento>
loop resultSet has next
«create»
«create»
evento: Evento
addEvento
listEventi
«create»
map: HashMap
creaMappaBiglietti(listEventi)
loop for each Evento
put(Evento.luogo,Evento.bigliettiVenduti)
setMapBiglietti(map)
setChanged(void)
notifyObservers
update

Statiche CRC Card:

Nome Classe: StatisticheController	**Package**: control
Superclasses	-
Subclasses	-
Responsabilità Aggiunge listener ai componenti della view Statistiche, crea un costruttore ed impone funzioni di aggiornamento per alcuni componenti della ricerca ed infine i Metodi comunicanti con il DAO	**Collaboratori** StatisticheModel StatisticheView EventoDAO

Nome Classe: StatisticheModel	**Package**: Model
Superclasses	Observable
Subclasses	-
Responsabilità Da le direttive per costruire i grafici per le statistiche	**Collaboratori**

Nome Classe: StatisticheView	**Package**: View
Superclasses	Observer
Subclasses	-
Responsabilità Questa Classe si occupa di rappresentare la schermata per visualizzare le statistiche gestendo i vari getter e di gestire il metodo update per i grafici presenti nella schermata	**Collaboratori** StatisticheModel

Classi Comuni
Qui sono descritti i CRC delle classi comuni ovvero quelle classi usate per descrivere piu class Diagram

Nome Classe: ControlloSintassi	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità si occupa dei controlli sulle stringe in partico-	**Collaboratori**

lare sulla sintassi del Codice fiscale, dell iban, e dell email.	

Nome Classe: CambiaStage	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Si occupa di cambiare gli Stage dell'applicazione	**Collaboratori** MenuPrincipaleView

Nome Classe: MenuPrincipaleController	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità Si occupa di cambiare *AnchorPane* nel form del Menu Principale, al verificarsi di eventi stabiliti, e carica le risorse FXML.	**Collaboratori** EventoDAO InserisciEventoView

Nome Classe: MenuPrincipaleView	**Package**: View
Superclasses	Initializable, ControlledStage
Subclasses	-
Responsabilità Questa classe si occupa di rappresentare la schermata di menu generale dove sono collegati tutti i punti salienti dell applicativo e con uno spazio per allocare le finestre per eseguire le diverse operazioni	**Collaboratori** Stage CambiaStage MenuPrincipaleController

Nome Classe: *GestoreQueryCerca*	**Package**: DB
Superclasses	-
Subclasses	-
Responsabilità	**Collaboratori**
Connessione al DB, prepara ed eseguire query per le varie ricerche clienti, eventi, addetti e altro	*UtilityDB*

Nome Classe: *UtilityDB*	**Package**: DB
Superclasses	-

Subclasses	-
Responsabilità	**Collaboratori**
Metodi statici per la connessione e la chiusura di connessione relative alla basi di dati Oracle	**-**

Nome Classe: GestoreQueryModificaElimina	**Package**: DB
Superclasses	-
Subclasses	-
Responsabilità	**Collaboratori**
Connessione al DB, prepara ed eseguire query per le operazioni di cancellazione di un cliente, evento, partecipanti e addetto oltre alle operzione di modifica per gli eventi	*UtilityDB*

Nome Classe: Persona	**Package**: Model
Superclasses	-
Subclasses	-
Responsabilità Crea un nuovo oggetto denominato persona con i suoi getter e setter per i suoi attributi	**Collaboratori**

Nome Classe: NoValidEventDataException	**Package**: Control
Superclasses	-
Subclasses	-
Responsabilità	**Collaboratori**

Nome Classe: MostraAlert	**Package**: View
Superclasses	-
Subclasses	-
Responsabilità Questa classe si occupa di dare un prototipo di messaggio di errore e di successo	**Collaboratori**