Object Design Document

Smart Planner

Riferimento	
Versione	0.4
Data	13/02/2024
Destinatario	Docente C. Gravino
Presentato da	Gennaro Foglia, Isabella Maria Sessa, Silvana Cafaro
Approvato da	

Team members

Nome	Acronimo	Informazione di contatto
Gennaro Foglia	GF	g.foglia9@studenti.unisa.it
Isabella Maria Sessa	IMS	i.sessa11@studenti.unisa.it
Silvana Cafaro	SC	s.cafaro7@studenti.unisa.it

Revision History

Data	Versione	Descrizione	Autori
20/01/2024	0.1	Prima stesura	Gennaro Foglia, Isabella Maria Sessa, Silvana Cafaro
21/01/2024	0.2	Packages, Class Interfaces, Class Diagram Ristrutturato	Gennaro Foglia, Isabella Maria Sessa, Silvana Cafaro
10/02/2024	0.3	Stesura finale	Gennaro Foglia, Isabella Maria Sessa, Silvana Cafaro
13/02/2024	0.4	Revisione finale	Gennaro Foglia, Isabella Maria Sessa, Silvana Cafaro

Indice

1. Introduzione

Dall'inizio della realizzazione ed implementazione di Smart Planner, il suo obiettivo è stato quello di rendere fruibile ai propri utenti un modo semplice e intuitivo di organizzare i propri impegni, senza stress e tempo perso dietro ad una classica agenda cartacea. Come dimostrato in molti studi, l'organizzazione in agenda dei propri impegni ed eventi rende più consapevoli ma soprattutto efficienti, ma quello che si promette di fare Smart Planner è un tassello più in alto. Infatti, l'applicazione permette di ridurre al minimo il fastidio di cercare un momento libero per studiare, fare la spesa o leggere un libro. Questo grazie al suo algoritmo di Intelligenza Artificiale, che organizza gli impegni dell'utente in base agli eventi già inseriti da questo nei propri calendari.

1.1 Linee Guida per la scrittura del codice

Per la definizione delle linee guida per la scrittura del codice si è fatto riferimento alla convenzione Java nota come Sun Java Coding Conventions.

Link alla documentazione ufficiale della convenzione

Java Sun: https://checkstyle.sourceforge.io/sun_style.html

1.2 Definizione, acronimi e abbreviazione

In questa sezione sono presenti le definizioni, gli acronimi e le abbreviazioni presenti in tutto il documento:

- **Package**: raggruppamento di classi, interfacce o file correlati;
- **Design pattern**: template di soluzioni per risolvere problemi ricorrenti, attraverso una struttura o un'organizzazione specifica del codice, per il riuso e la flessibilità;
- **Interfaccia**: insieme di firme dei metodi che la classe mette a disposizione;
- **ODD**: Object Design Document;

1.3 Riferimenti e link utili

In questa sezione sono presenti dei riferimenti a documenti esterni che facilitano la lettura dell'ODD:

- Requirements Analysis Document;
- System Design Document;
- Test Plan;

2. Package

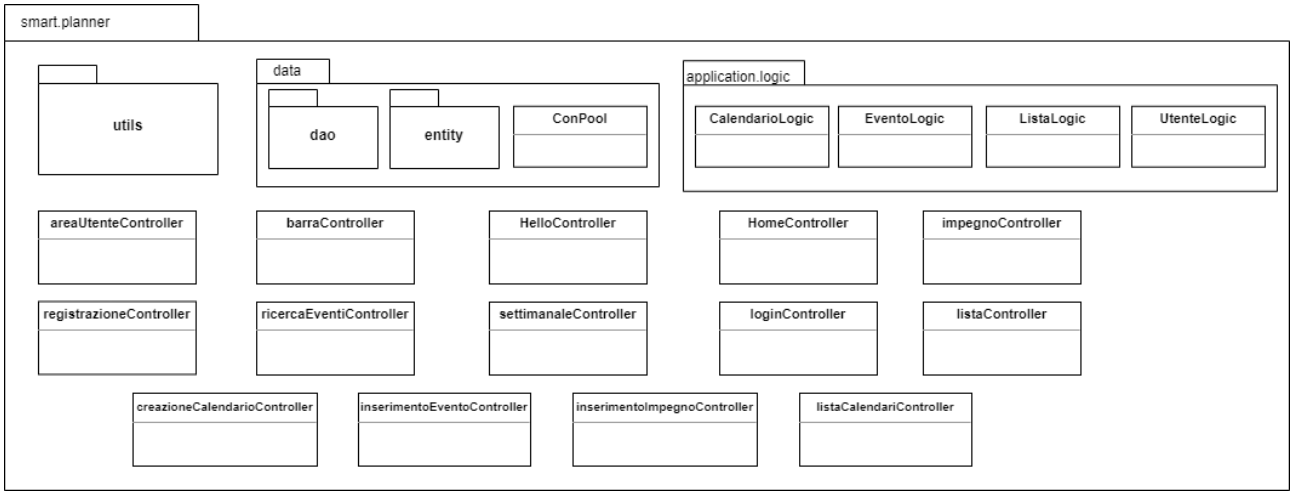
In questa porzione del documento si mostra in quali packages è diviso il sistema, questo in base a quanto definito nel documento System Design. La suddivisione in packages attuale è motivata da scelte architetturali prese e segue la struttura di directory standard definita dallo strumento di gestione Maven.

- **.idea**
- **.mvn**: contiene i file di configurazione per Maven.
- **src**: contiene i file sorgenti.
    - **main**
        - **java**
            - **org.genisilv.smartplanner**: contiene le classi riguardanti la logica dell'applicazione e le classi per l'accesso al database.
                - **application.logic**
                - **data**
                - **utils**
        - **resources**: contiene i file fxml e le immagini.
    - **Test**: contiene i file per i testing.
        - **Java**: contiene le classi per l'implementazione del testing.
- **Target**: contiene i file prodotti dal sistema di build di Maven.

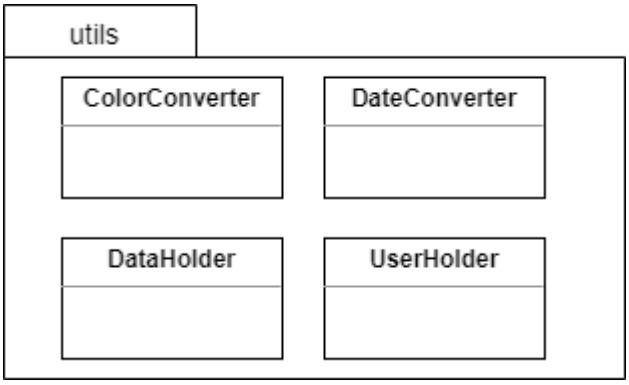
Package di Smart Planner

In questo paragrafo presentiamo la struttura principale di Smart Planner, composta da 3 package principali:

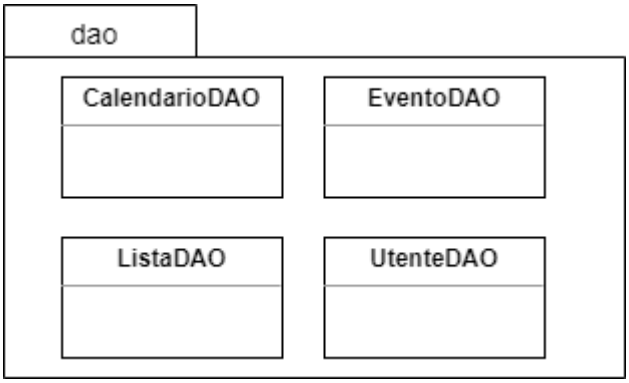
- **Package application.logic**: contiene i sottosistemi per la gestione dell'utente, del calendario, dell'evento, della lista.
- **Package data**: contiene le classi per la gestione del Database.
- **Package utils**: contiene classi che sono usate da diverse componenti del sistema.



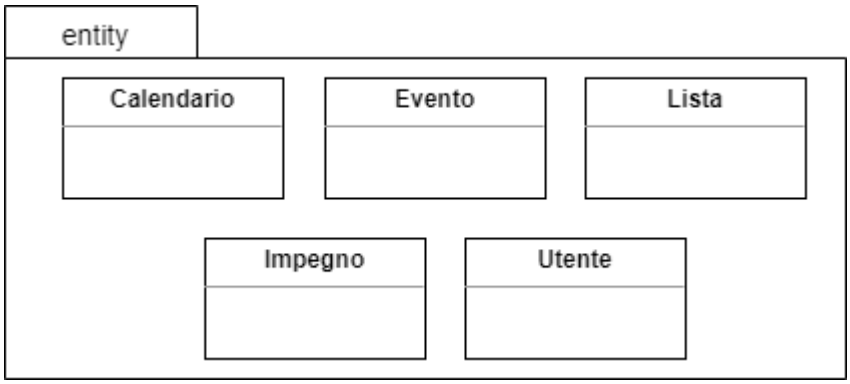
Package Utils



Package dao



Package entity



3. Class Interfaces

In questo paragrafo sono presenti le interfacce di ogni classe.

1. Package Calendario

Nome Classe	CalendarioLogic
Descrizione	La seguente classe consente di gestire le operazioni relativamente ai calendari dell'applicazione.
Metodi	+creaCalendario(String nome, Color color): void +cercaCalendario(String nome): Calendario +returnCalendari(): ArrayList<Calendario> +aggiungiEvento(int codiceEvento, int codiceCalendario): void
Invariante di classe	/

Nome metodo	+creaCalendario(String nome, Color color): void
Descrizione	Il seguente metodo gestisce la richiesta di creazione di un calendario nell'applicazione.
Pre-condizione	/
Post-condizione	/
Nome metodo	+cercaCalendario(String nome): Calendario
Descrizione	Il seguente metodo gestisce la ricerca di un calendario all'interno dell'applicazione.
Pre-condizione	/
Post-condizione	/
Nome metodo	+returnCalendari(): ArrayList<Calendario>
Descrizione	Il seguente metodo restituisce tutti i calendari dell'utente.
Pre-condizione	/
Post-condizione	/
Nome metodo	+aggiungiEvento(int codiceEvento, int codiceCalendario): void
Descrizione	Il seguente metodo consente l'aggiunta di un evento all'interno del calendario.
Pre-condizione	/
Post-condizione	/

2. Package Evento

Nome Classe	EventoLogic
Descrizione	La seguente classe consente di gestire le operazioni relativamente agli eventi dell'applicazione.
Metodi	+aggiungiEvento(String nome, String descrizione, GregorianCalendar data, String oraInizio, String oraFine, Color colore, boolean notifiche, int periodicita): int +ricercaEvento(String nome, String email): ArrayList<Evento> -checkName(String nome) : boolean -lunghezza (String descrizione) : boolean -date (GregorianCalendar data) : boolean -checkTimeE(Stringa oraI, Stringa oraF) : boolean
Invariante di classe	/

Nome metodo	+aggiungiEvento(String nome, String descrizione, GregorianCalendar data, String oraInizio, String oraFine, Color colore, boolean notifiche, int periodicita): int
Descrizione	Il seguente metodo consente la creazione di un evento.
Pre-condizione	checkName == true && Lunghezza == true && Date == true && checkTimeE == true
Post-condizione	EventoDAO.doSaveEvento(evento) > 0
Nome metodo	+ricercaEvento(String nome, String email): ArrayList<Evento>
Descrizione	Il seguente metodo gestisce la ricerca di un evento all'interno dell'applicazione.
Pre-condizione	/
Post-condizione	/
Nome metodo	-checkName(String nome) : boolean
Descrizione	Il seguente metodo controlla il formato del nome.
Pre-condizione	La stringa nome rispetta la regex: "[^a-zA-Z0-9]"
Post-condizione	/
Nome metodo	-lunghezza(String descrizione) : boolean
Descrizione	Il seguente metodo controlla la lunghezza della descrizione.
Pre-condizione	La stringa descrizione è minore di 200 caratteri.
Post-condizione	/
Nome metodo	-date(GregorianCalendar data) : boolean
Descrizione	Il seguente metodo controlla la data dell'evento.
Pre-condizione	La data dell'evento non deve essere antecedente alla data dell'inserimento.
Post-condizione	/
Nome metodo	-checkTimeE(Stringa oraI, Stringa oraF) : boolean
Descrizione	Il seguente metodo controlla che gli orari di inizio e fine siano corretti.

Pre-condizione	oraFine>oraInizio || if oraFine==oraInizio then minutiFine>minutiInizio
Post-condizione	/

3. Package Lista

Nome Classe	ListaLogic
Descrizione	La seguente classe consente di gestire le operazioni relativamente alla lista dell'applicazione.
Metodi	+creaLista (Color colore, ArrayList<Impegno> impegni): void +creaLista(Color colore): void +aggiungiImpegno(String nomeImpegno, int durataImpegno, int prioritaImpegno): int +svuotaLista(): void +cancellaImpegno(int codiceImpegno): void -checkName(String nome) : boolean
Invariante di classe	/

Nome metodo	+creaLista (Color colore, ArrayList<Impegno> impegni): void
Descrizione	Il seguente metodo consente di gestire la creazione della lista inserendo anche gli impegni al suo interno.
Pre-condizione	/
Post-condizione	/
Nome metodo	+creaLista(Color colore): void
Descrizione	Il seguente metodo consente di gestire la creazione della lista.
Pre-condizione	/
Post-condizione	/
Nome metodo	+aggiungiImpegno(String nomeImpegno, int durataImpegno, int prioritaImpegno): int
Descrizione	Il seguente metodo consente di gestire l'aggiunta di un impegno alla lista.
Pre-condizione	checkName == true
Post-condizione	ListaDAO.doAddImpegno(impegno) > 0
Nome metodo	+svuotaLista(): void
Descrizione	Il seguente metodo consente di svuotare la lista, eliminando tutti gli impegni al suo interno.
Pre-condizione	/
Post-condizione	/
Nome metodo	+cancellaImpegno(int codiceImpegno): void
Descrizione	Il seguente metodo consente di cancellare un impegno all'interno della lista.
Pre-condizione	/
Post-condizione	/
Nome metodo	-checkName(Stringa nome) : boolean
Descrizione	Il seguente metodo controlla il formato del nome.
Pre-condizione	La stringa nome rispetta la seguente regex: "[^a-zA-Z0-9]"
Post-condizione	/

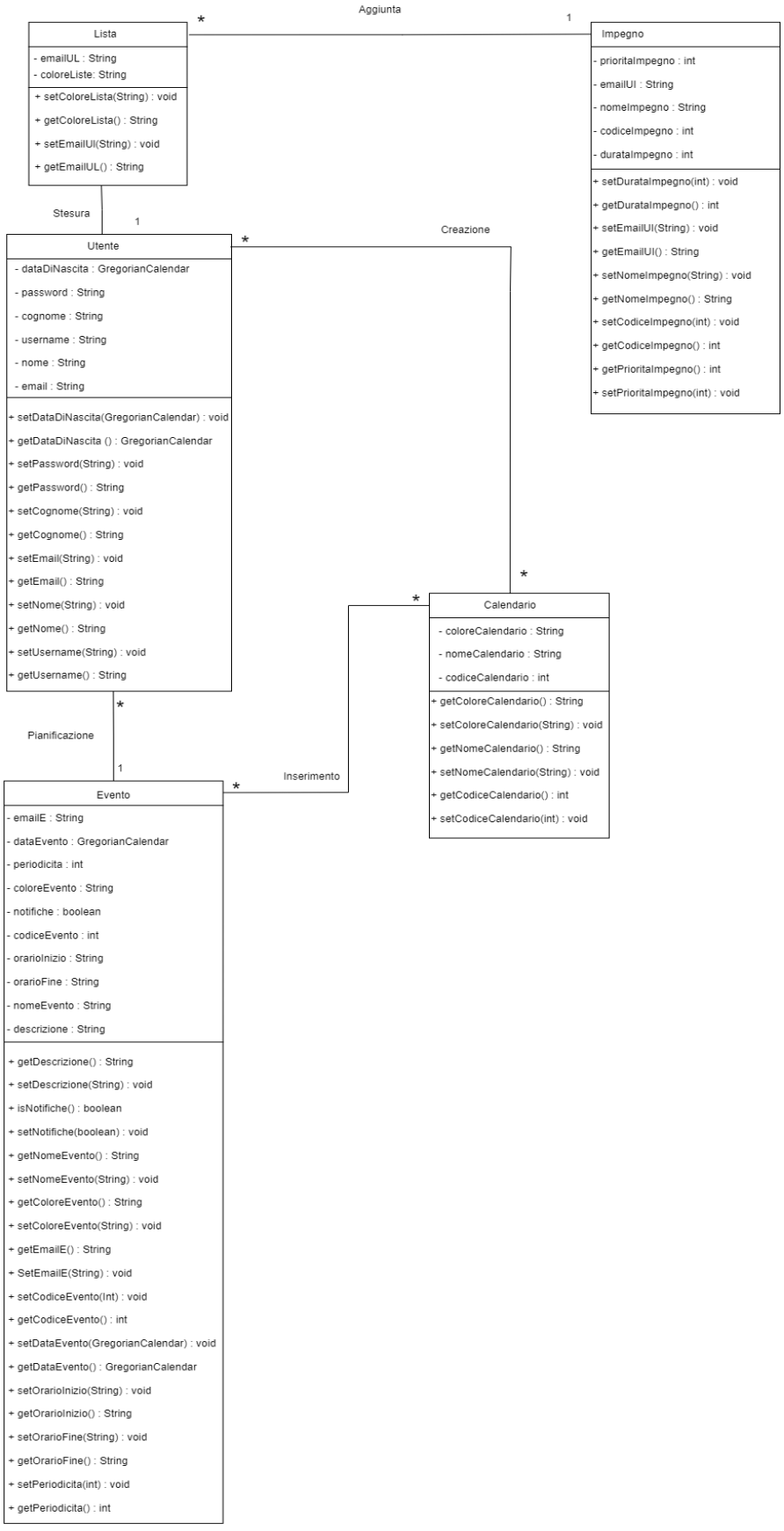
4. Package Utente

Nome Classe	UtenteLogic
Descrizione	La seguente classe consente di gestire le operazioni relativamente agli utenti dell'applicazione.
Metodi	+registrazione(String email, String password, String nome, String cognome, GregorianCalendar nascita, String username): void +login(String email, String password): int +cancellaUtente(): void +returnUtenteByEmail(String email): Utente +logout(): void +returnLoggedInUser(): Utente -checkName(String nome) : boolean -date(GregorianCalendar nascita) : boolean -checkEmail(String email) : boolean -formato(String username) : boolean -lunghezza(String password) : boolean
Invariante di classe	/

Nome metodo	+registrazione(String email, String password, String nome, String cognome, GregorianCalendar nascita, String username): void
Descrizione	Il seguente metodo consente la registrazione di un utente all'applicazione.
Pre-condizione	checkName == true && checkName == true && Date == true && checkEmail == true && Formato == true && lunghezza==true
Post-condizione	
Nome metodo	+login(String email, String password): int
Descrizione	Il seguente metodo consente il login di un utente all'applicazione.
Pre-condizione	checkEmail == true
Post-condizione	Utente != null
Nome metodo	+cancellaUtente(): void
Descrizione	Il seguente metodo consente di eliminare permanentemente l'account di un utente.
Pre-condizione	/
Post-condizione	/
Nome metodo	+returnUtenteByEmail(String email): Utente
Descrizione	Il seguente metodo consente la ricerca di un utente tramite email.
Pre-condizione	/

Post-condizione	/
Nome metodo	+logout(): void
Descrizione	Il seguente metodo consente di effettuare il logout di un utente dall'applicazione.
Pre-condizione	/
Post-condizione	/
Nome metodo	+returnLoggedInUser(): Utente
Descrizione	Il seguente metodo gestisce l'utente loggato.
Pre-condizione	/
Post-condizione	/
Nome metodo	-checkName(String nome) : boolean
Descrizione	Il seguente metodo controlla il formato del nome.
Pre-condizione	La stringa nome rispetta la seguente regex: "[^a-zA-Z]"
Post-condizione	/
Nome metodo	-date(GregorianCalendar nascita) : boolean
Descrizione	Il seguente metodo controlla la data di nascita dell'utente.
Pre-condizione	Nascita >= 16 anni.
Post-condizione	/
Nome metodo	-checkEmail(String email) : boolean
Descrizione	Il seguente metodo controlla il formato dell'email
Pre-condizione	La stringa email rispetta la seguente regex: "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}$"
Post-condizione	/
Nome metodo	-formato(String username) : boolean
Descrizione	Il seguente metodo controlla il formato dello username.
Pre-condizione	La stringa username rispetta la seguente regex: "[^a-zA-Z0-9]"
Post-condizione	/
Nome metodo	-lunghezza(String password) : boolean
Descrizione	Il seguente metodo controlla la lunghezza della password.
Pre-condizione	La lunghezza della password è >= 8.
Post-condizione	/

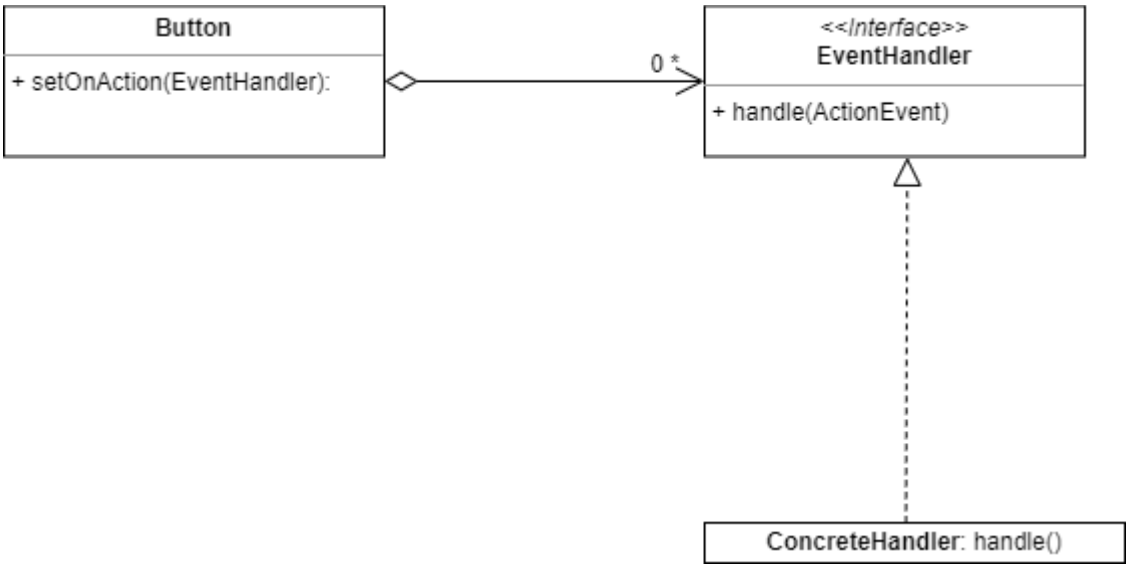
4. Class Diagram Ristrutturato



5. Elementi di Riuso

5.1 Design Pattern Usati

In questo paragrafo è descritto il design pattern utilizzato per la realizzazione del progetto.
Il design pattern utilizzato è Observer, data la necessità di aggiornare elementi dell'interfaccia in modo dinamico.
Nella pagina in cui viene visualizzato un calendario, alla pressione del bottone corrispondente a un giorno della settimana vengono aggiornati: la posizione di un elemento puramente grafico dell'interfaccia, la lista degli eventi a schermo per visualizzare quelli presenti a quel giorno e la data presente in alto per allinearla a quella del giorno.



5.2 Componenti COTS

È stato usato lo strumento SceneBuilder, che consente di creare rapidamente e in modo intuitivo interfacce per un'applicazione JavaFX.