Gestore Portineria

Progetto di Programmazione e Modellazione ad oggetti

Michael Tomassini
Matricola 302059

17 Marzo 2023

Indice

Capitolo 1
Analisi

Mi pongo come obiettivo quello di realizzare un'applicazione gestionale per la portineria di un generico edificio. Il compito del portiere richiede responsabilità e precisione; egli deve poter fornire e attingere rapidamente alle informazioni sullo stato dell'edificio. La presenza di un’applicazione capace di organizzare e semplificare questo lavoro ha lo scopo di ridurre al minimo gli errori umani dovuti alla disorganizzazione o alla presenza di eccessive scartoffie. L'applicazione supporterà l'autenticazione tramite credenziali personali, la gestione delle chiavi dell'edificio, la comunicazione di note/circolari tra i dipendenti della portineria e la creazione/rimozione di nuove credenziali di autenticazione nonché delle varie chiavi dell’edificio.

1.1 Requisiti

Requisiti funzionali:

- L’applicazione richiederà il possesso di credenziali di accesso per essere utilizzata, ovvero username e password, uniche per ogni portiere.
- Sarà possibile la creazione di nuovi utenti, nonché la loro rimozione.
- Sarà possibile visualizzare e modificare il registro delle chiavi dell'edificio.
- Sarà possibile inserire molteplici note e comunicazioni destinate alla lettura da parte degli altri dipendenti, nonché recuperarne e visualizzarne di vecchie.
- I dati vengono salvati in un file, che viene caricato all’avvio dell’applicazione e aggiornato durante l’utilizzo.

Requisiti non funzionali:

- L’applicazione deve essere intuitiva e facile da usare.
- Se il database risultasse assente, verrà creato. Verrà poi inserito un account con username "admin" e password "admin", avente lo scopo di permettere il primo utilizzo dell'applicazione.

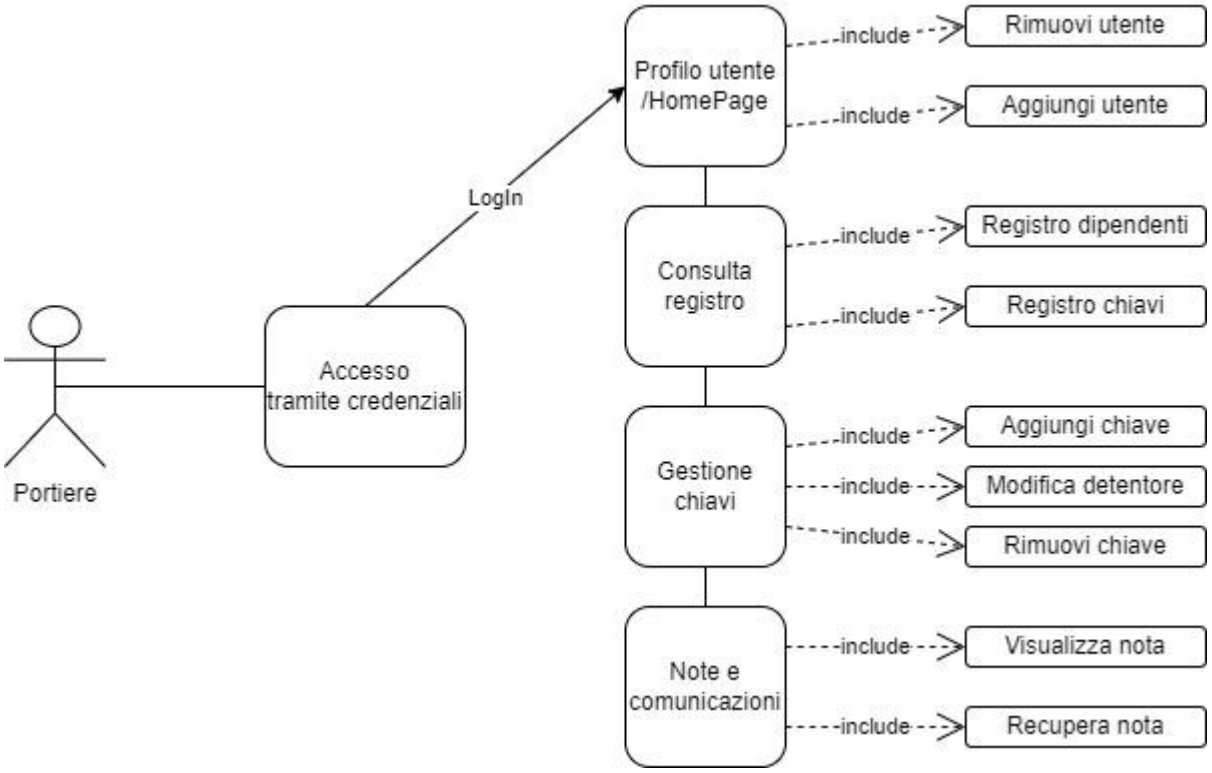

Figura 1.1: Diagramma dei casi d'uso

1.2 Modello del dominio

Il modello del dominio dell'applicazione è composto da molteplici entità che interagiscono tra loro per gestire le diverse funzionalità offerte.
DataManager si occupa della comunicazione dell'applicazione con il database, rappresenta un registro generico, il quale viene esteso e specializzato dalle interfacce KeysManager, NoteManager ed EmployeesManager.

- KeysManager si occupa del registro delle chiavi dell'edificio. Ogni chiave è rappresentata dall'interfaccia Key, la sua implementazione contiene informazioni sul detentore e un'etichetta univoca.

- EmployeesManager si occupa della gestione dei dipendenti della portineria. Attraverso questa interfaccia, è possibile creare, aggiornare e rimuovere dipendenti, nonché visualizzare le loro informazioni personali. L'implementazione dell'interfaccia Employee rappresenta un singolo dipendente, che può accedere all'applicazione utilizzando le proprie credenziali personali. Il processo di autenticazione è gestito dalla classe LogInController.

- LogInController gestisce il processo di autenticazione dei dipendenti all'interno dell'applicazione. Si occupa di interagire con l'interfaccia grafica, ricevere le credenziali inserite dall'utente e verificare se queste sono corrette attraverso l'impiego di EmployeesManager.

- NoteManager gestisce le note e le comunicazioni tra i dipendenti della portineria. Fornisce funzionalità per aggiungere, recuperare e visualizzare le comunicazioni tra i dipendenti, facilitando lo scambio di informazioni all'interno della portineria. L'implementazione dell'interfaccia Note rappresenta una singola nota o comunicazione, identificata dalla propria data di creazione.

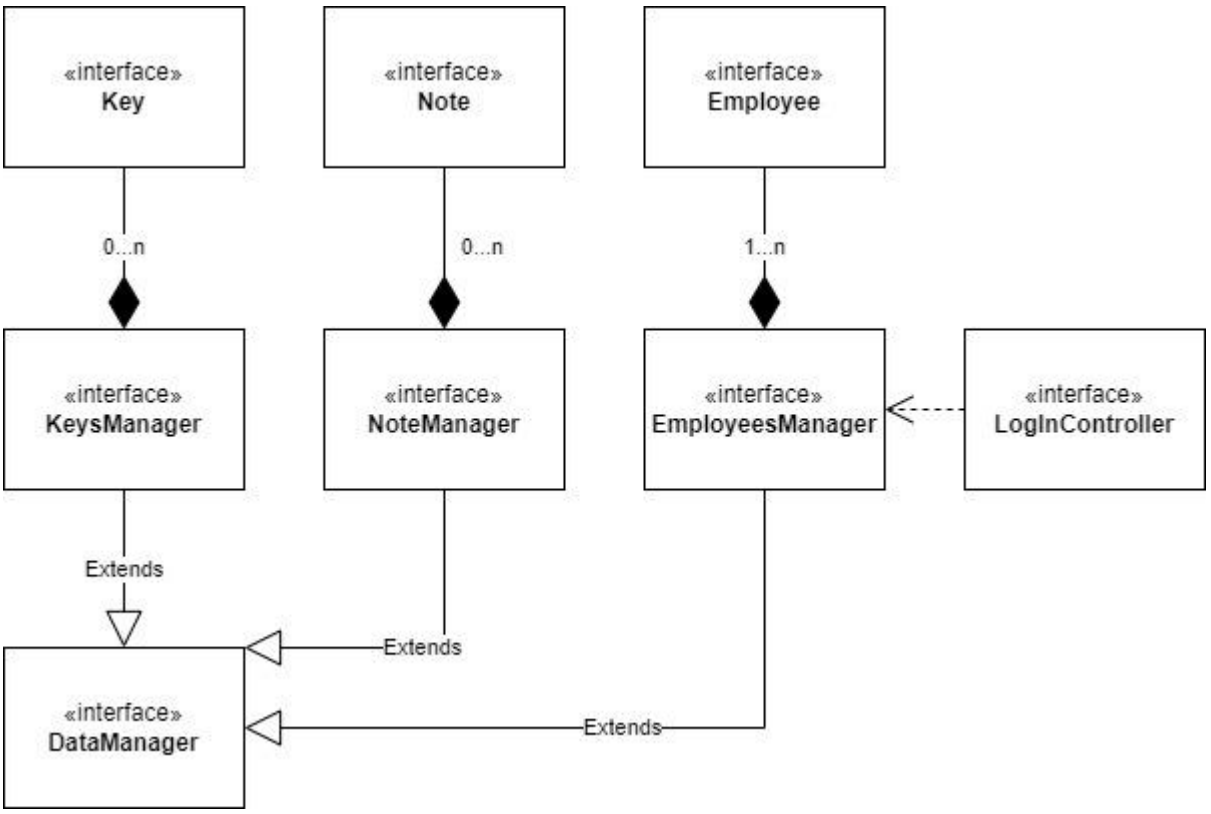

Figura 1.2: Schema del modello del dominio

Capitolo 2
Design

2.1 Architettura

L'architettura del programma è stata progettata seguendo il pattern architetturale Model-View-Controller (MVC), che è stato scelto per la sua capacità di separare in modo efficiente la logica dell'applicazione dalla sua interfaccia grafica. L'obiettivo principale del progetto è fornire un'applicazione user-friendly per gestire e organizzare le informazioni relative ai dipendenti, alle chiavi e alle note. Per raggiungere questo scopo, sono state adottate diverse tecnologie e librerie, come JavaFX per l'interfaccia utente e SQLite per la gestione del database, che saranno discusse nelle sezioni seguenti.

2.1.1 Model

Il Model si occupa dell'accesso ai dati, gestisce il database e consente la comunicazione tra quest'ultimo e l'applicazione. Il suo compito è quello di mostrare al controller lo stato dell'applicazione in ogni momento.
Per creare il Model, ho definito delle interfacce per ogni elemento facente parte delle funzionalità enunciate dall'analisi del progetto, cioè note (Note), dipendenti (Employee) e chiavi (Key).
Suddette interfacce definiscono il contratto per gli oggetti di tipo Note, Employee e Key, rappresentando rispettivamente le note create all'interno dell'applicazione, i dipendenti e le chiavi. Esse vengono utilizzate da apposite specializzazioni dell'interfaccia DataManager, rispettivamente EmployeesManagerImpl, KeysManagerImpl e NoteManagerImpl.
Queste classi manager seguono il pattern Data Access Object (DAO), che fornisce un'astrazione tra la logica dell'applicazione e l'accesso ai dati. In questo modo, è possibile separare le responsabilità e facilitare eventuali modifiche al modo in cui i dati vengono memorizzati o recuperati.
EmployeesManagerImpl, KeysManagerImpl e NoteManagerImpl, su richiesta del Controller, si occuperanno di gestire eventuali modifiche alle liste di cui sono composte e, sulla base dei metodi messi a disposizione, effettueranno gli opportuni mutamenti al database. Queste classi utilizzano la connessione al database fornita dal metodo getConnection() dell'interfaccia DataManager<T> per eseguire operazioni di creazione, lettura, aggiornamento e cancellazione sugli oggetti Key, Employee e Note.
Grazie a queste implementazioni, le liste sono mantenute aggiornate e sincronizzate con il database, garantendo un'efficiente gestione delle informazioni all'interne dell'applicazione.

2.1.2 View

La View si occupa di visualizzare i dati contenuti nel Model e gestire l'interazione con l'utente attraverso i metodi forniti al Controller.
Per l'implementazione dell'interfaccia grafica è stata utilizzata la libreria JavaFX in concomitanza alla creazione di file FXML, cioè markup XML che definiscono il layout di una GUI permettendo l'implementazione del pattern MVC.

Questi file FXML consentono quindi di separare la logica dell'applicazione dalla sua presentazione, seguendo il principio di separazione delle responsabilità.

La View è suddivisa in due interfacce grafiche principali:

- L'interfaccia dedicata al LogIn, che permette all'utente di effettuare l'accesso all'applicazione
- L'interfaccia dedicata alla pagina principale, ovvero all'utilizzo effettivo dell'applicazione.

Quest'ultima interfaccia grafica è suddivisa in quattro schermate intercambiabili, ovvero Utente, Registro, Chiavi e Note, ognuna dedicata alle proprie funzionalità di competenza. Inoltre è presente una terza più basilare interfaccia grafica dedicata esclusivamente all'inserimento di credenziali atte alla creazione di un nuovo utente.

I file FXML vengono utilizzati in combinazione con i Controller, che gestiscono gli eventi generati dall'interazione dell'utente con la GUI. In questo modo, la View interagisce con il Model e il Controller in modo strutturato e coerente.

2.1.3 Controller

Il Controller è la componente responsabile di gestire le interazioni dell'utente con l'interfaccia grafica, comunicando al Model le informazioni sulle modifiche effettuate. Quando il Model ha completato l'elaborazione delle richieste, il Controller aggiornerà la View in modo che questa possa essere visualizzata coerentemente. In questo modo, il Controller si comporta da intermediario tra la View e il Model, assicurando che l'interazione dell'utente con l'applicazione sia gestita in modo efficace e coerente con la logica del pattern MVC. Seguendo la suddivisione della View, ho deciso di dividere il controller in due classi principali, identificate come LogInController e MainPageController, e una terza classe denominata AddEmployeeController.

2.1.4 Tecnologie e librerie utilizzate

Durante lo sviluppo del progetto, sono state valutate diverse alternative per l'implementazione delle tecnologie e delle librerie. Tra queste, JavaFX è stato scelto al posto di Swing per l'interfaccia utente, principalmente per la sua capacità di creare interfacce più moderne e la sua maggiore flessibilità nella creazione di layout personalizzati.
Per quanto riguarda la gestione del database, SQLite è stata scelta come soluzione per memorizzare e gestire i dati degli oggetti Employee, Key e Note. SQLite è un'opzione leggera e serverless che si integra facilmente con l'applicazione e riesce ad offrire prestazioni sufficienti per il progetto in questione.

2.2 Design dettagliato

Gestione del database e dei dati:

Il progetto utilizza SQLite come database per memorizzare e gestire le informazioni degli oggetti Employee, Key e Note. Le informazioni relative ad essi vengono immagazzinate in omonime tabelle e accedute all’avvio dell’applicazione. L'interfaccia DataManager<T> ricopre un ruolo fondamentale nella gestione dell'accesso dei dati, fornendo il metodo getConnection() che permette di stabilire una connessione con il database SQLite attraverso il campo statico url. Le classi KeysManagerImpl, EmployeeManagerImpl e NoteManagerImpl estendono l'interfaccia DataManager<T> e utilizzano la connessione al database per eseguire operazioni di creazione, lettura, aggiornamento e rimozione.

KeysManagerImpl si specializza per la gestione degli oggetti KeyImpl.

NoteManagerImpl si specializza per la gestione degli oggetti NoteImp,

EmployeesManagerImpl si specializza per la gestione degli oggetti EmployeeImpl.

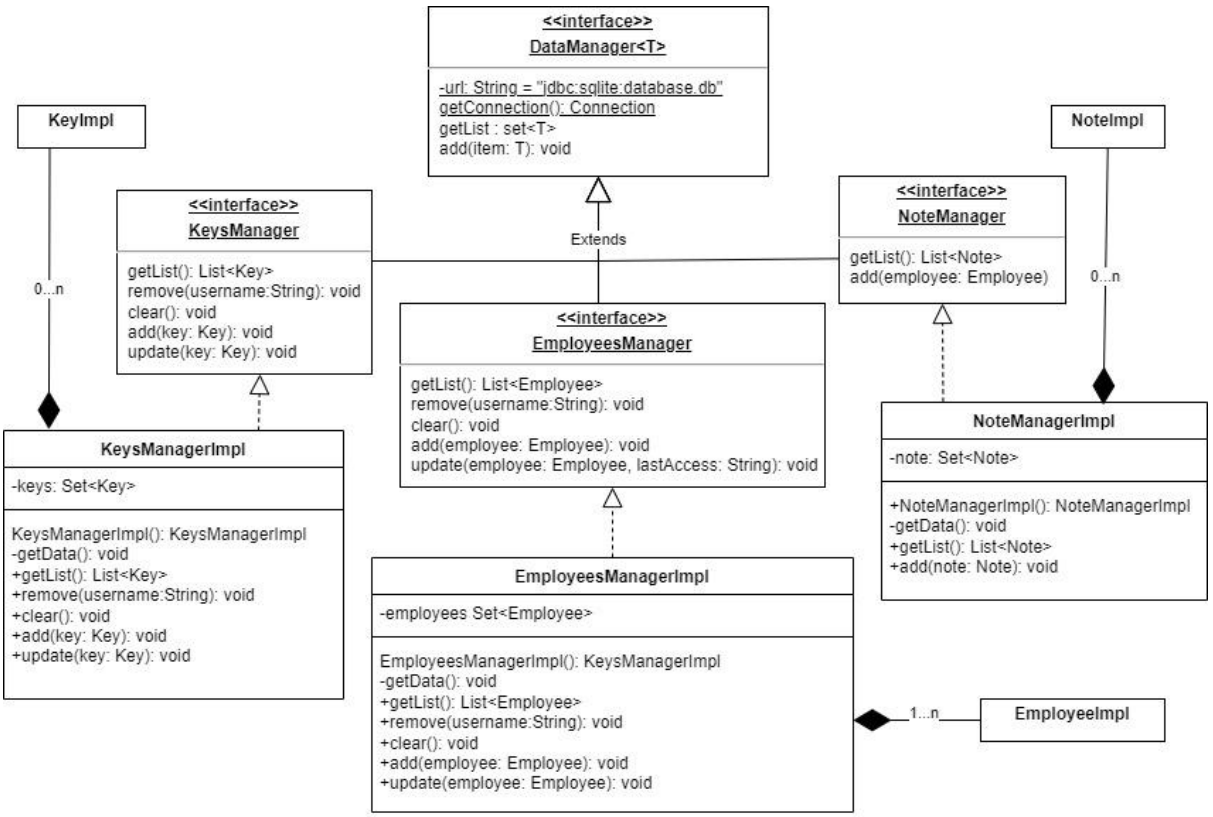

Figura 2.1: UML Model e gestione dei dati

L'interfaccia Key, implementata in KeyImpl, definisce il contratto per gli oggetti Key.

L'interfaccia Note, implementata in NoteImpl, definisce il contratto per gli oggetti Note,

L'interfaccia Employee, implementata in EmployeeImpl, definisce il contratto per gli oggetti Employee, che rappresentano i dipendenti all'interno del sistema.
Un oggetto Employee ha un nome, un cognome, un username, una data dell'ultimo accesso e uno stato di accesso (variabile booleana positiva quando l'utente è connesso).

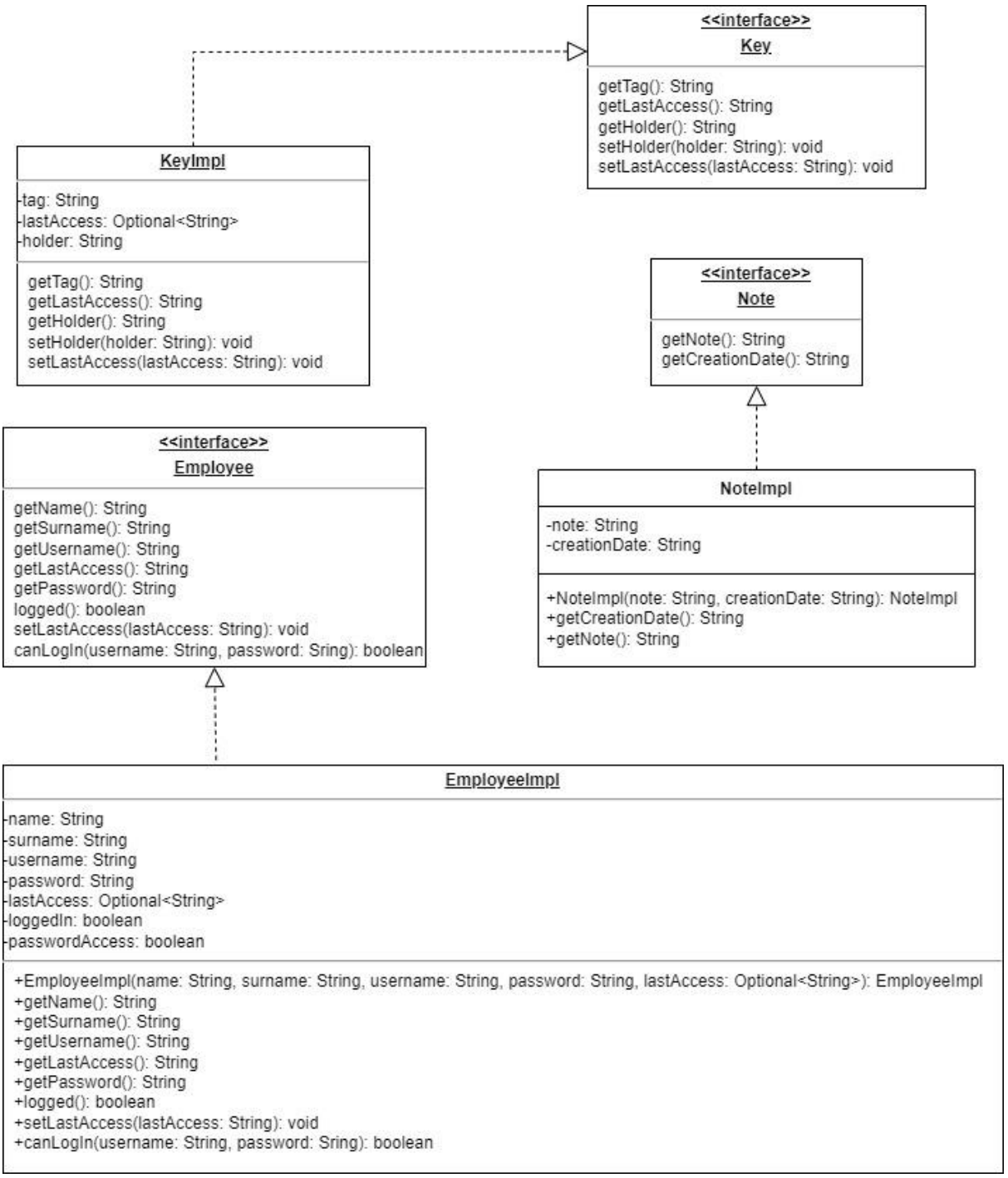

Figura 2.2: Employee, key, note

Gestione dell’autenticazione:

L'applicazione necessita di un sistema di autenticazione per garantire che solo gli utenti autorizzati possano accedervi. A tale scopo ho deciso di implementare un meccanismo di login che utilizza un controller (LogInController) per gestire l'autenticazione.
Il controller collabora con LogIn.fxml e la classe EmployeesManagerImpl.
Il metodo userLogIn() viene chiamato quando l'utente cerca di accedere all'applicazione e verifica se le credenziali inserite, presenti nei campi FXML denominati username e password, corrispondono a quelle di un dipendente esistente.
Se l'autenticazione ha successo verrà chiamato il metodo changeScene() per passare alla schermata principale dell'applicazione “Mainpage.fxml”; altrimenti, viene mostrato un messaggio di errore.

Gestione della schermata principale:

Dopo l'autenticazione, l'applicazione deve fornire un'interfaccia utente che permetta di interagire con diverse funzionalità, come la gestione dei dipendenti, delle chiavi e delle note. Ho dedicato a tale scopo la scrittura del MainPageController, che coordina le diverse sezioni. MainPageController utilizza oggetti manager (EmployeesManager, KeysManager, NotesManager) per gestire le liste di dipendenti, chiavi e note, e interagisce con i vari elementi dell'interfaccia grafica per visualizzare e modificare i dati.
Le sezioni e i relativi più rilevanti metodi impiegati per la risoluzione del problema, presenti nel suddetto Controller, sono:

- Sezione utente: metodo userCall() per visualizzare informazioni sull'utente corrente. addUser() e removeUser() per la possibilità di aggiungere o rimuovere utenti.
- Sezione registro: metodo registerCall() per visualizzare il registro dei dipendenti e delle chiavi.
- Sezione gestione delle chiavi: metodi per aggiungere, rimuovere e modificare il possessore delle chiavi, rispettivamente keyCall(), removeKey() e modifyKey().
- Sezione note del personale: metodi per visualizzare e salvare le note, rispettivamente noteCall() e saveTextArea().
- Logout: il metodo logOut() esegue il logout dell'utente corrente e ritorna alla schermata di login (LogIn.fxml).

Per ottenere questo risultato vengono impiegati 4 oggetti AnchorPane, rispettivamente: utente, registro, keys e note. Ognuno contenente il layout della propria specifica sezione e vengono resi visibili/invisibili coerentemente all’interazione dell’utente con l’interfaccia grafica. Per esempio, se l'utente preme il pulsante per accedere alla schermata delle note, il metodo noteCall() viene chiamato. Questo metodo rende invisibili le altre schermate e imposta la visibilità della schermata delle note a true.

Creazione di nuove credenziali di accesso:

Il controller AddEmployeeControl gestisce l'aggiunta di un nuovo dipendente nell'applicazione. È composto dai seguenti elementi principali:

- Campi FXML: vari elementi dell'interfaccia utente come TextField per inserire il nome, cognome, nome utente e password del nuovo dipendente, e un campo Text per visualizzare eventuali messaggi di errore.
- Oggetto manager: EmployeesManager viene utilizzato per l'aggiunta di un nuovo dipendente, tramite il proprio metodo add().

Il metodo addUser() viene chiamato quando si clicca sul pulsante"Aggiungi utente" per aggiungere un nuovo dipendente. Se uno o più campi sono vuoti, mostra un messaggio di errore "Errore: dati mancanti!".

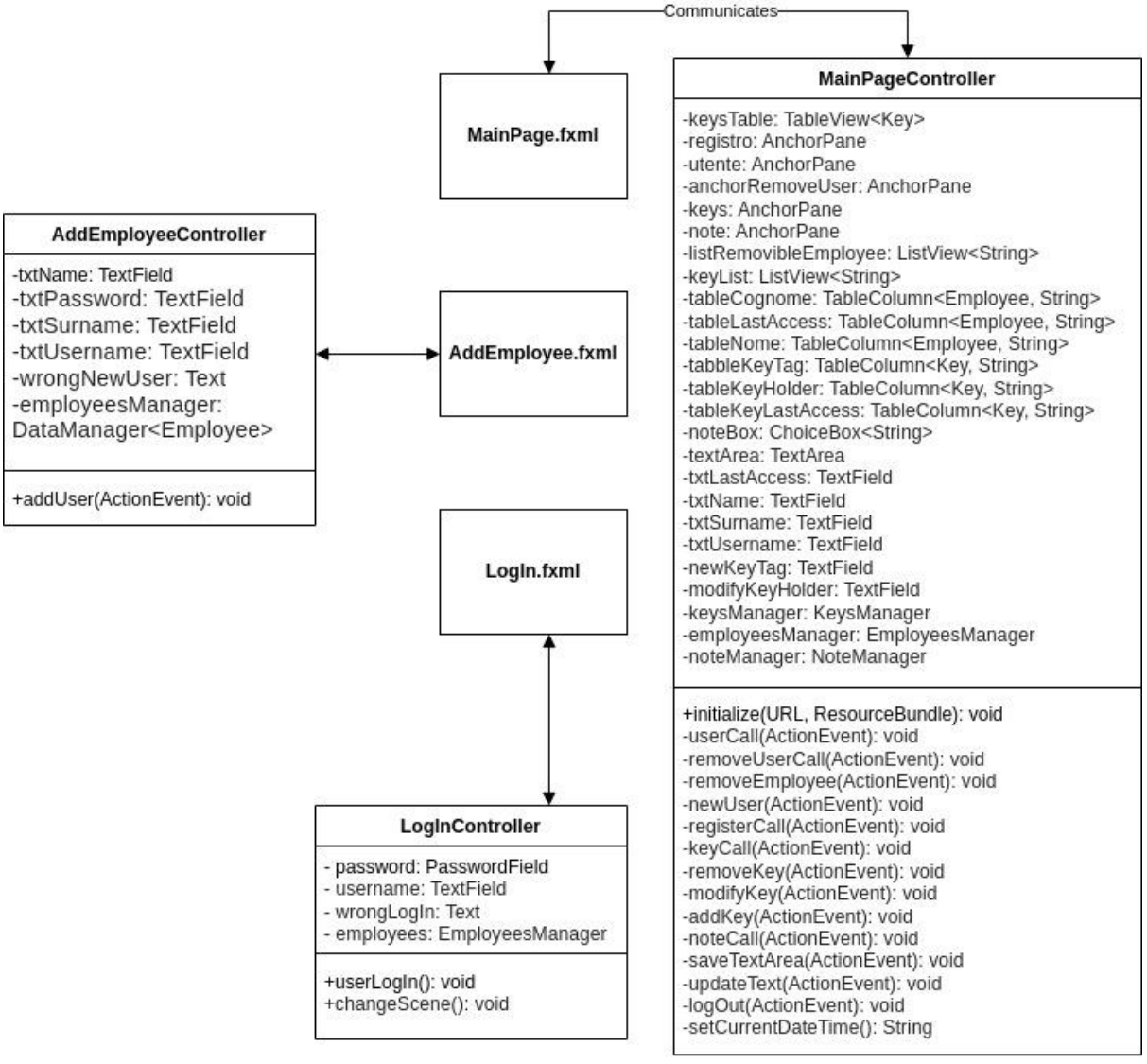

Figura 2.3: View Controller

Rappresentazione delle interfacce grafiche:

Le interfacce grafiche ".fxml" lavorano insieme al controller per fornire l'interazione con l'applicazione. In questo caso i file ".fxml" sono layout XML che definiscono la struttura e l'aspetto dell'interfaccia utente.
Nel progetto, ogni controller è associato a un file ".fxml" specifico, e la relazione tra i due è stabilita attraverso gli elementi FXML e i metodi del controller.
Gli elementi FXML nei file ".fxml" sono collegati ai campi corrispondenti nel controller tramite l'annotazione @FXML.
Questo permette al controller di utilizzare gli elementi dell'interfaccia utente.
i metodi del controller sono associati agli eventi dell'interfaccia utente, come il clic di un pulsante o la selezione di un elemento in un elenco.
Questa associazione viene effettuata tramite gli attributi "onAction" nei file ".fxml" che fanno riferimento ai metodi corrispondenti del controller.
Quando l'utente interagisce con l'interfaccia, i metodi del controller vengono chiamati per gestire l'evento e aggiornare l'interfaccia o i dati di conseguenza.

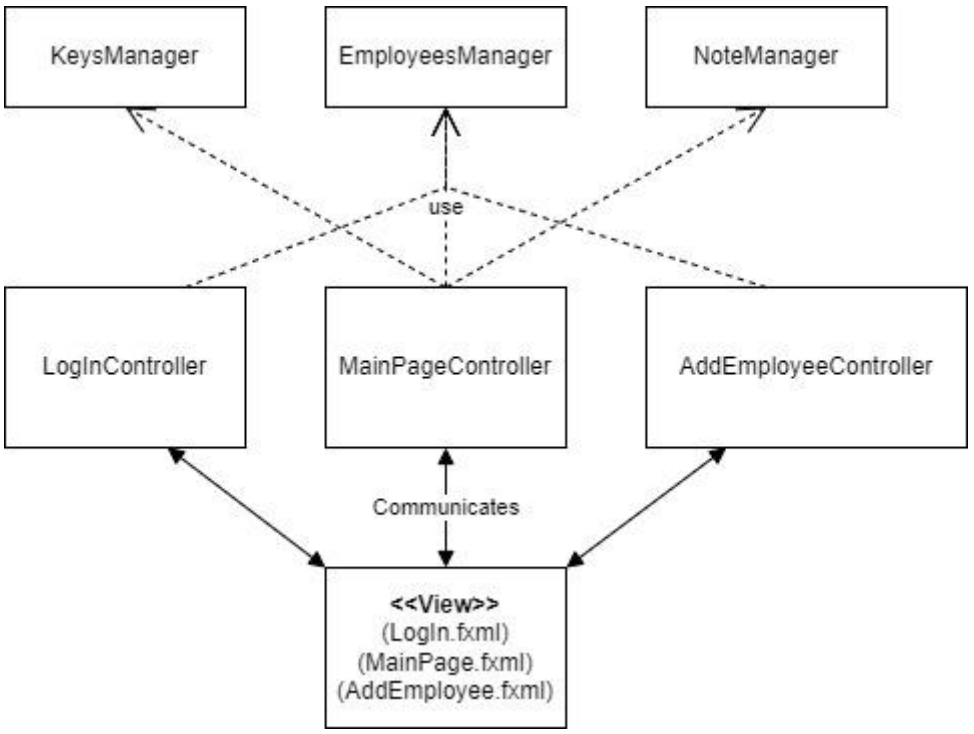

Figura 2.4: Rappresentazione Model View Controller

Capitolo 3
Sviluppo

3.1 Testing automatizzato

Durante il processo di sviluppo, ho integrato le diverse componenti del progetto, assicurandomi che funzionassero correttamente insieme. Ho sviluppato e testato ogni singola parte prima di passare alla successiva. In particolare, ho prestato attenzione all'integrazione tra il Model e i Controller JavaFX, e alla corretta comunicazione tra le classi Manager e le loro interfacce. Ho effettuato test funzionali manuali e automatizzati attraverso l'impiego del framework JUnit 5. Quest'ultimo mi ha permesso ad ogni nuovo metodo o feature aggiunta ad una classe la possibilità di verificarne il funzionamento in modo indipendente e mirato. I test prodotti sono stati:

- DatabaseTest
- KeysTest
- EmployeesTest
- NoteTest

3.2 Metodologia di lavoro

Una volta individuata l'idea per l'applicazione, la fase iniziale dello sviluppo ha portato alla luce interrogativi sulle funzionalità che essa avrebbe dovuto fornire all'utilizzatore.
Lo studio dei casi d'uso è stato il primo passo, vista la natura del problema e l'analisi degli obiettivi prefissati, ho immaginato quali potessero essere gli strumenti fondamentali e necessari all'utilizzo dell'applicazione nel contesto del problema designato.
A questo punto, pensando l'applicazione come un insieme di molteplici strumenti, ho prediletto un approccio modulare, dove ogni componente (o strumento) potesse essere rimosso o sostituito senza inficiare il resto del programma.
Ciò mi ha permesso di individuare più chiaramente il modello del dominio, che ha rappresentato il punto di partenza dal quale poter iniziare la stesura del codice.
Ho fatto uso di Eclipse come ambiente di sviluppo, iniziando a scrivere le interfacce facenti parte del Model, e le loro conseguenti implementazioni.
Ad ogni implementazione ho affiancato dei test utili alla manutenibilità di queste nel corso dello sviluppo.
Git si è rivelato uno strumento fondamentale, permettendomi di testare nuove funzionalità tenendo traccia dei cambiamenti apportati di volta in volta al progetto, nonché poter lavorare senza essere vincolato ad una singola postazione sfruttando l'utilizzo del servizio di hosting GitHub.
Non essendoci altri sviluppatori, ho optato per l'impiego di un singolo branch.
Visto l'utilizzo di GitHub, in special modo per semplificare la gestione delle dipendenze, ho trovato utile l'impiego dello strumento di gestione dei progetti Maven.

Procedendo con la realizzazione, e concentrandomi sulla GUI, ho impiegato Scene Builder per definire l'interfaccia grafica dell'applicazione.
Scene Builder è uno strumento utile per la creazione di file ".fxml" utilizzati in progetti JavaFX.
Da un punto di vista del design pattern MVC, utilizzando Scene Builder per definire l'interfaccia utente, viene separato il codice che gestisce l'aspetto dell'interfaccia (la View) dalla logica del Model e del Controller, semplificando il processo di sviluppo e manutenzione del codice.
Non essendoci altri sviluppatori coinvolti nel progetto, tutte le parti del codice sono state sviluppate in autonomia.
Tuttavia, ho consultato risorse online e mi sono documentato per approfondire diversi aspetti specifici del linguaggio Java e delle librerie utilizzate, come JavaFX, Maven, driver JDBC per l'utilizzo di SQLite.

3.3 Note di sviluppo

Di seguito verranno esposte le caratteristiche avanzate di Java che sono state impiegate per sviluppare le classi implementate nel progetto:

Progettazione con generici:

- Uso di generici bounded nella classe DataManager<T> .

Uso di Stream e Optional:

- Utilizzo di Stream per scovare, manipolare e trasformare le raccolte di oggetti nelle classi manager e controller.
- Uso di Optional per gestire valori che potrebbero essere null.