Documentazione progetto Ingegneria del Software

Vincenzo Sanzone, VR471513

July 5, 2023

Contents

1 Requisiti ed interazione utente-sistema

1.1 Specifiche casi d'uso

Il sistema a cui fa riferimento la seguente documentazione è quello relativo alla gestione dei Passaporti. Gli attori previsti sono 3 i cittadini, gli addetti della questura e l'admin. Gli addetti verranno inseriti da parte dell'admin, mentre i cittadini potranno registrarsi in maniera autonoma.

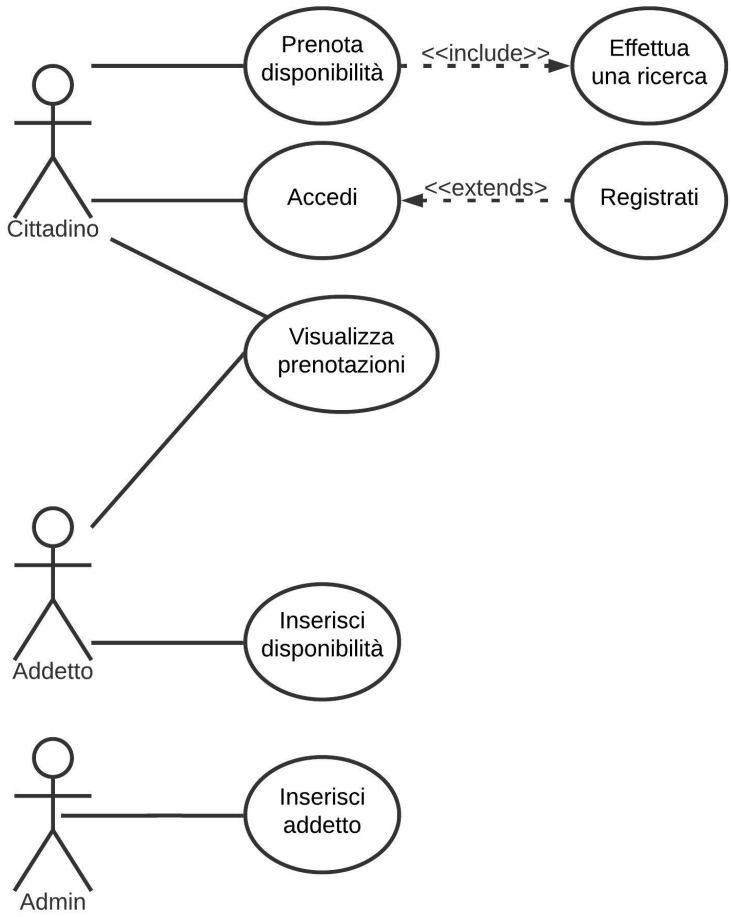


Figure 1: Nell'immagine abbiamo i casi d'uso

Per una migliore leggibilità delle specifiche dei casi d'uso, sono stati volutamente omessi il caso in cui l'utente decida di tornare alla schermata precedente.

1.1.1 Casi d'uso cittadino

Caso d'uso: Accesso.
Attori: Cittadino.
Precondizioni: 1. L'utente deve aver scelto l'interfaccia relativa al cittadino. 2. Il cittadino deve essersi registrato.
Sequenza degli eventi: 1. Il cittadino è introdotto all'interfaccia d'accesso. 2. Il cittadino inserisce le credenziali.
Postcondizioni: 1. Il cittadino accede all'interfaccia di prenotazione.
Sequenza alternativa 1: 1. In qualunque momento il cittadino può accedere all'interfaccia di registrazione.
Postcondizioni: 1. Il cittadino visualizza l'interfaccia di registrazione.

Ovviamente, il sistema verificherà le credenziali, se non dovessero essere corrette veranno richieste.

Caso d'uso: Registrazione.
Attori: Cittadino.
Precondizioni: 1. L'utente deve trovarsi nell'interfaccia "Registra cittadino".
Sequenza degli eventi: 1. Il cittadino inserisce i campi richiesti.
Postcondizioni: 1. Il cittadino verrà registrato. 2. Il cittadino accede all'interfaccia di prenotazione.

In questo caso, il sistema si assicurerà che le informazioni inserite dal cittadino siano coerenti tra di loro.

Caso d'uso: Prenota una disponibilità.
Attori: Cittadino.
Precondizioni: 1. L'utente deve trovarsi nell'interfaccia di prenotazione. 2. L'utente deve aver effettuato una ricerca
Sequenza degli eventi: 1. Il cittadino sceglie la disponibilità che preferisce.
Postcondizioni: 1. Il cittadino avrà prenotato quella disponibilità.

Il sistema effettuerà dei controlli assicurandosi che la prenotazione sia compatibile con le informazioni che ha.

Caso d'uso: Effettua una ricerca.
Attori: Cittadino.
Precondizioni: 1. L'utente deve trovarsi nell'interfaccia di prenotazione.
Sequenza degli eventi: 1. Il cittadino inserisce i filtri (opzionale). 2. Il cittadino sottomette la ricerca.
Postcondizioni: 1. Il cittadino potrà visionare le disponibilità, coerenti con le ricerche.

Il sistema controllerà che la ricerca abbia senso.

Caso d'uso: Visualizza prenotazioni.
Attori: Cittadino, Addetto
Precondizioni: 1. L'utente deve essersi autenticato.
Sequenza degli eventi: 1. L'utente accede nella sezione prenotazione. 2. L'utente può visionare le disponibilità prenotate e non.
Postcondizioni: Nessuna

1.1.2 Casi d'uso addetto

Per l'addetto della questura abbiamo un unico caso d'uso proprio.

Caso d'uso: Inserisci disponibilità.
Attori: Addetto.
Precondizioni: 1. L'addetto deve essersi autenticato.
Sequenza degli eventi: 1. L'addetto inserisce i campi richiesti.
Postcondizioni: 1. La disponibilità verrà registrata.

I campi inseriti dall'addetto devono essere coerenti.

1.1.3 Casi d'uso admin

L'admin è un attore che interagisce poco con il sistema, in quanto il suo unico compito è di inserire gli addetti della questura.

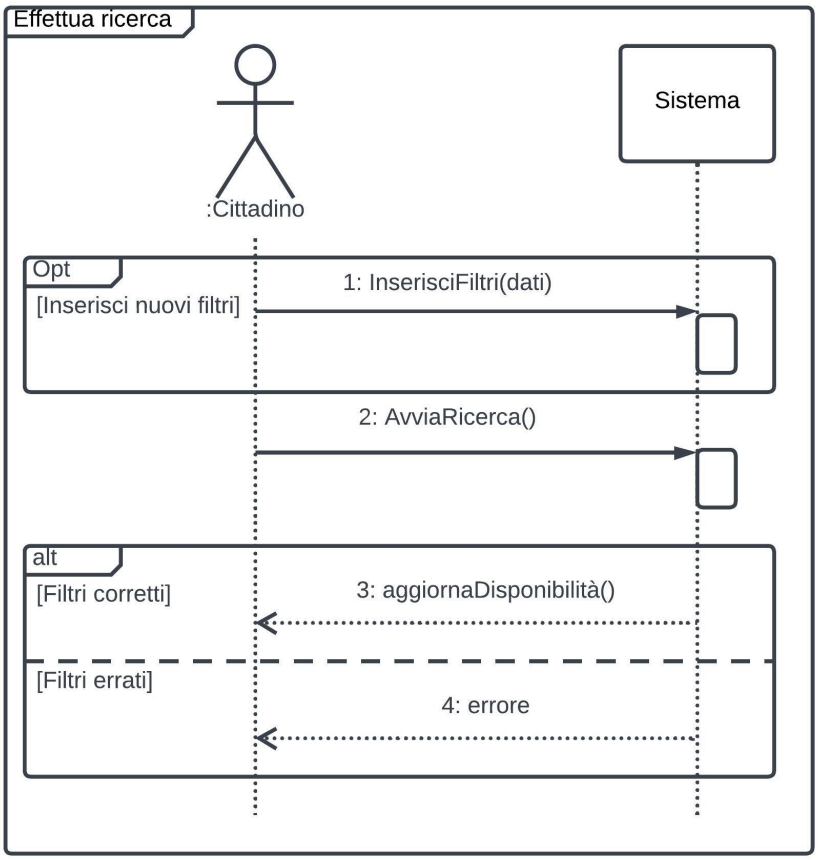
Caso d'uso: Inserisci addetto.
Attori: Admin.
Precondizioni: 1. L'admin deve essersi autenticato.
Sequenza degli eventi: 1. L'admin inserisce le credenziali.
Postcondizioni: 1. L'addetto verrà registrato.

1.2 Diagrammi di sequenza

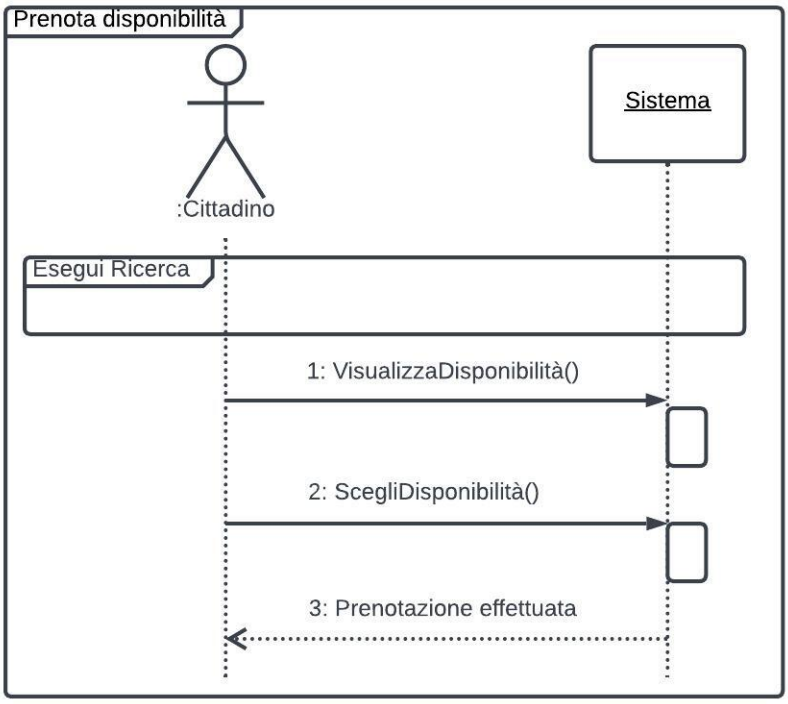
In questa sezione troviamo i diagrammi di sequenza dei casi d'uso più importanti.

1.2.1 Cittadino

Per il cittadino riportiamo quelli relativi alla ricerca e prenotazione.

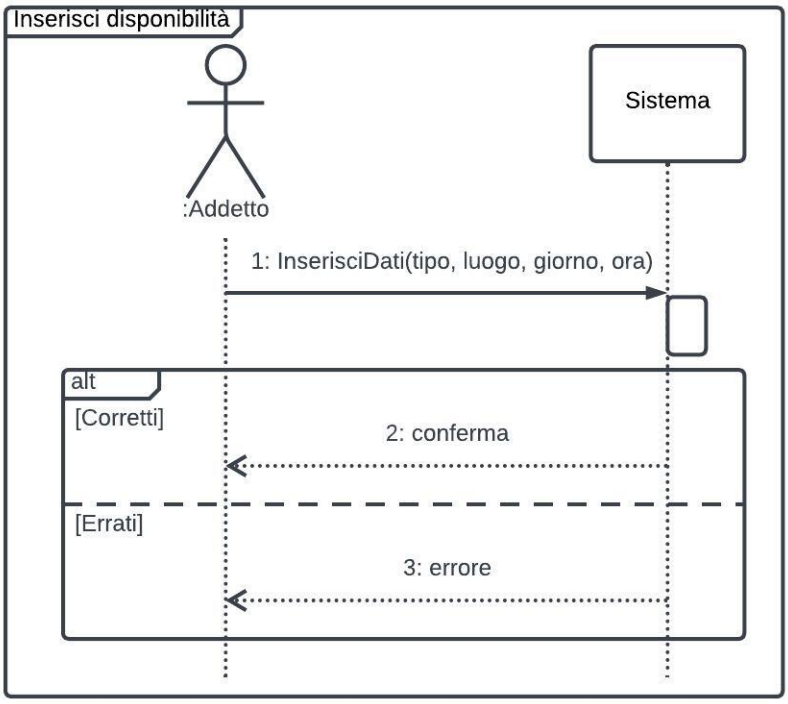


La prenotazione utilizza il blocco precedentemente mostrato per l'esecuzione della ricerca



1.2.2 Addetto

Per l'addetto riportiamo l'inserimento della disponibilità.



Il diagramma di sequenza, per il caso d'uso "inserimento dell'addetto", è sostanzialmente uguale al precedente.

1.3 Diagrammi di attività

Di seguito vengono riportate i diagrammi di attività del cittadino, addetto ed admin. I diagrammi rappresentano un'unica istanza, per una leggibilità migliore.

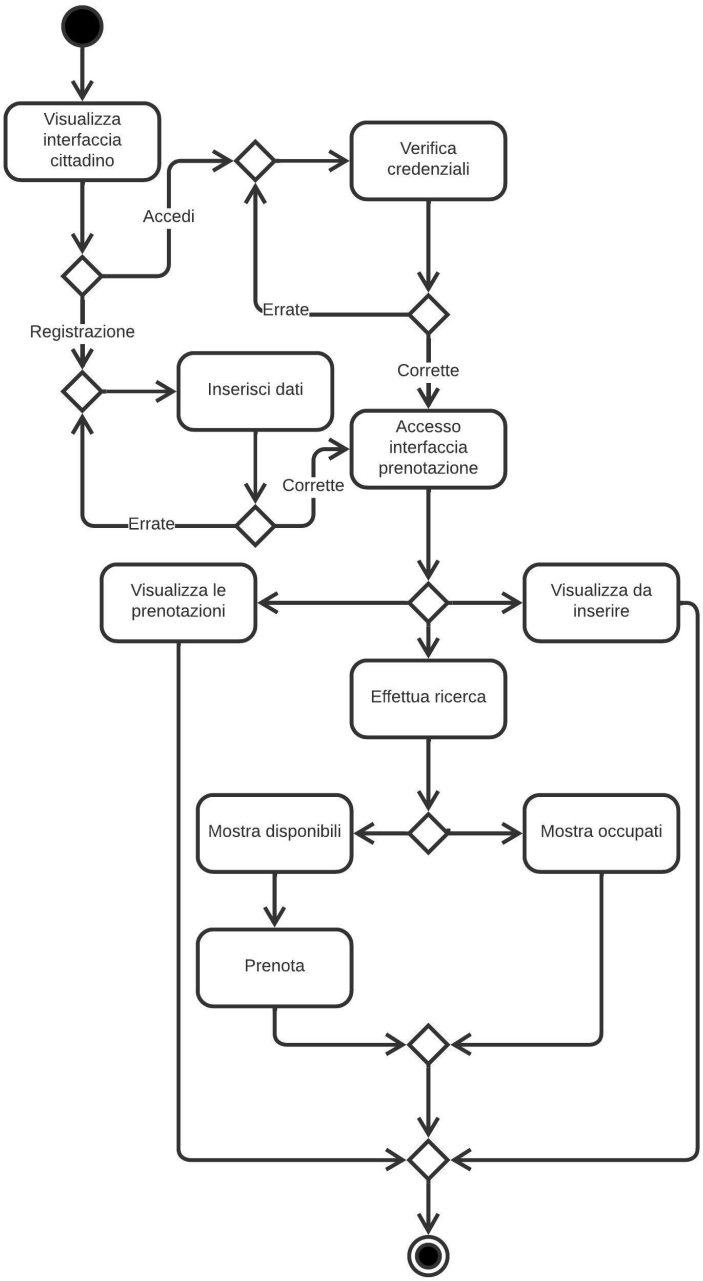


Figure 2: Attività cittadino

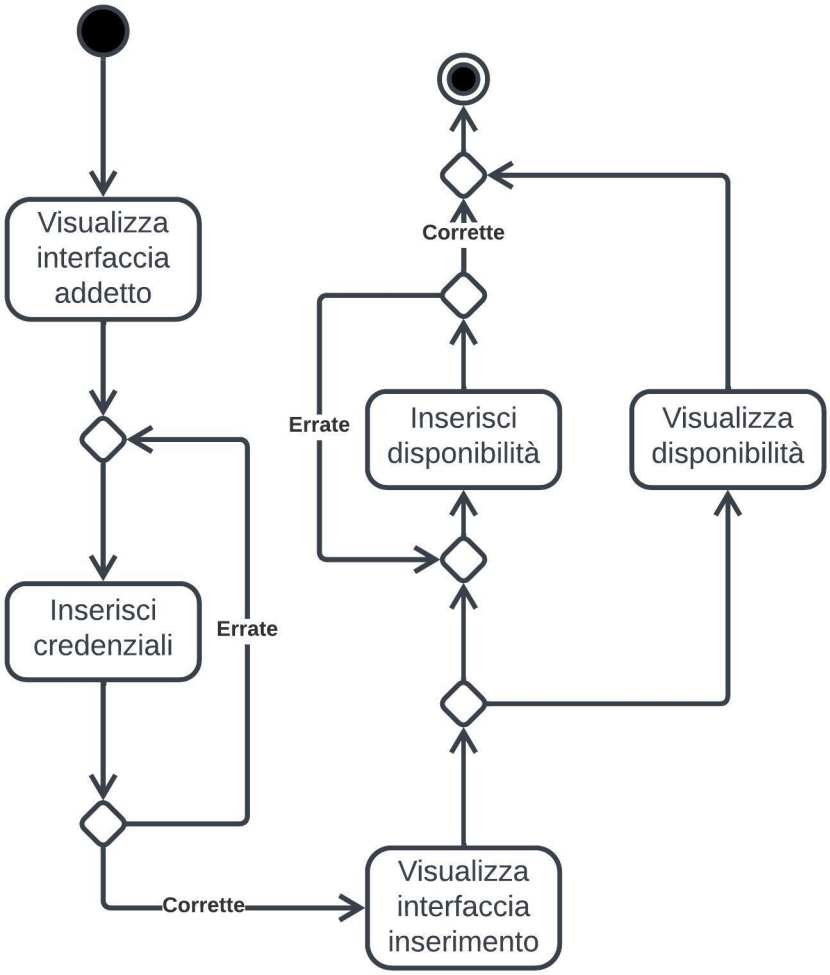


Figure 3: Attività addetto

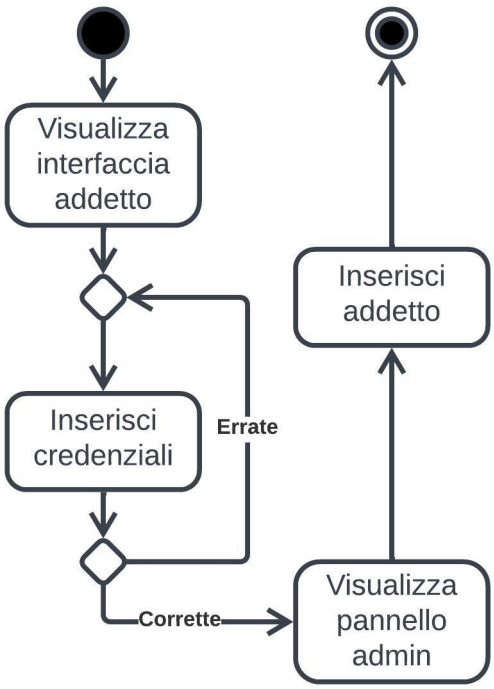


Figure 4: Attività admin

2 Sviluppo del software

La creazione del software ha seguito lo sviluppo incrementale, così facendo è stato suddiviso il lavoro in progettazione, implementazione e validazione.
Durante la fase di progettazione si ci è concentrati sul raggiungere gli obiettivi imposti dai casi d'uso, potendo così descrivere i primi test, semplici e non, su cui l'implementazione si doveva basare.
Durante la seconda fase, si è cercato di rappresentare al meglio le funzionalità richieste dalle specifiche. Quindi, si sono usati i risultati ottenuti dalla progettazione, e sono stati inseriti sotto forma di codice. Durante questa fase sono state apportate modifiche, migliorie e aggiunte di ulteriori test, per coprire casi precedentemente sfuggiti.
Nella terza parte si sono svolti tutti i test precedentemente trovati, e si sono cercate combinazioni che potessero portare il software a risultati non attesi. Però approfondiremo più nel dettaglio questa sezione successivamente.
Una volta finita la fase di validazione, si è ripreso il ciclo di sviluppo, passando ad un altro caso d'uso.

2.1 Pattern architetturali usati

Il sistema è stato prodotto attraverso l'uso del linguaggio Java. Per l'interfaccia utente si è deciso di usare JavaFX, con il supporto di SceneBuilder per la creazione dei file FXML, mentre per l'accesso ai database la scelta è ricaduta su SQLite.
Con questa nota, si comprende per quale motivo è stato adottato il pattern architetturale MVC, dove il progetto viene suddiviso in 3 parti che comunicano tra di loro.

- **Model:** regola l'accesso al database, inserendo ed estrapolando i dati da esso. I dati vengono scambiati in entrata o in uscita con il controller.

- **View:** rappresenta ciò che visualizza l'utente. Quest'ultimo interagisce con la vista, aspettandosi un cambiamento, che avverrà attravvero la logica implementata nel controller.

- **Controller:** gestisce il comportamento ad ogni singola azione dell'utente. Funge da intermediario tra vista e model permettendo ad entrambi di aggiornarsi.

Tutte le classi di Model utilizzano il pattern singleton, in questo modo il controller userà un'unica istanza del Model.

2.2 Diagramma delle classi

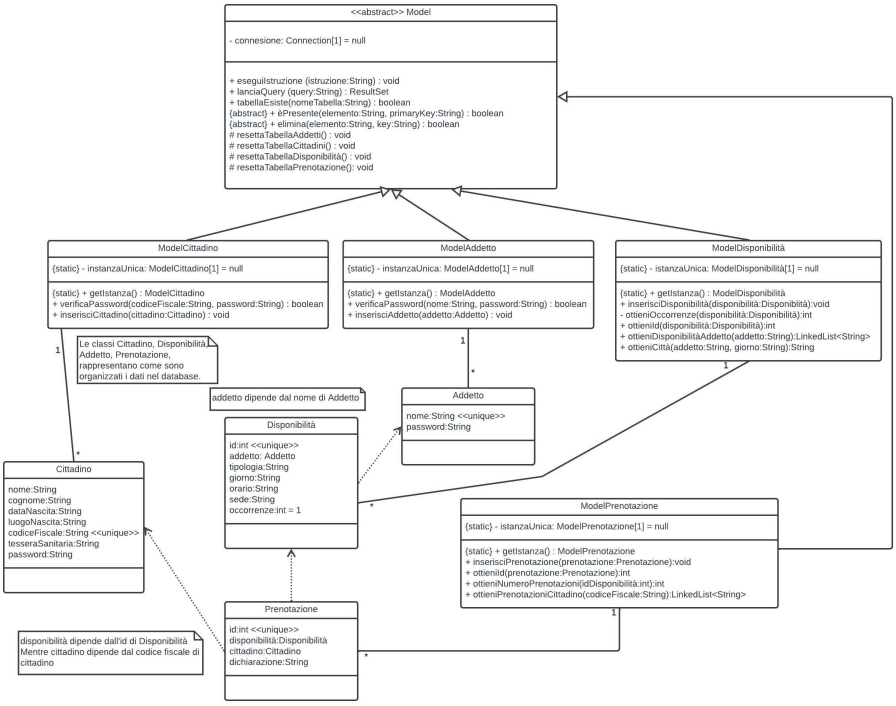


Figure 5: Diagramma delle classi di Model

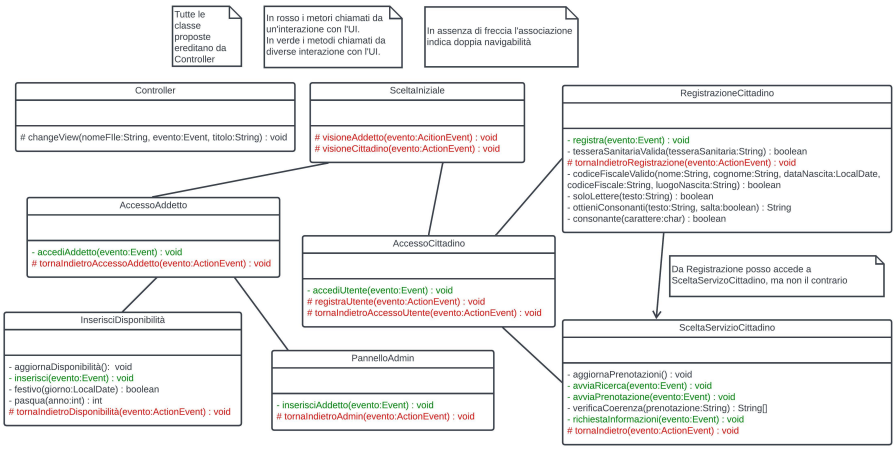


Figure 6: Diagramma delle classi di Controller

Le classi Controller permettono di cambiare ciò che vede l'utente, sotto la giusta sequenza di eventi.
Riportiamo adesso una breve spiegazione dei metodi delle varie classi. Per maggiori dettagli, si può visionare il JavaDoc.

- Model:
  - **eseguiIstruzione:** lancia un'istruzione SQL.
  - **lanciaQuery:** lancia una query SQL.
  - **tabellaEsiste:** verifica se una determinata tabella è presente nel database.
  - **èPresente:** verifica se un elemento esiste all'interno di una tabella.
  - **elimina:** elimina un determinato elemento da una tabella.
  - **resettaTabellaX:** elimina e ricrea la tabella X. (X può essere Addetti, Cittadini, Disponibilità, Prenotazione).
  - **getIstanza:** non è un metodo di Model, ma delle sue sottoclassi, per cui lo riportiamo una sola volta qui. Ritorna un istanza del modello ricercato.
- ModelCittadino:
  - **verificaPassword:** controlla se la password del codice fiscale è corretta.
  - **inserisciCittadino:** aggiunge le informazioni relative al cittadino, nella sua tabella.
- ModelAddetto: come per ModelCittadino, con la differenza che si fa riferimento alla tabella addetti.
- ModelDisponiblità:
  - **inserisciDisponilità:** inserisce una disponibilità nella sua tabella.
  - **ottieniOccorrenze:** restituisce il numero di occorrenze di una determinata disponibilità.
  - **ottieniId:** restituisce l'id univoco di una determinata disponibilità.
  - **ottieniDisponibilitàAddetto:** restituisce la lista di tutte le disponibilià di un addetto.
  - **ottieniCittà:** restituisce la città in cui si trova un addetto in un determinato giorno.
- ModelPrenotazione:
  - **inserisciPrenotazione:** inserisce una prenotazione nella sua tabella.
  - **ottieniId:** restituisce l'id univoco di una determinata prenotazione.
  - **ottieniNumeroPrenotazione:** restituisce il numero di volte in cui una disponibilità è stata prenotata.

- **ottieniPrenotazioniCittadino:** restituisce la lista delle prenotazioni di un cittadino.

- Controller: ha il metodo **changeView** che permette di cambiare schermata. Tutte le sotto classi di controller hanno il metodo **tornaIndietro** che permette di tornare alla schermata precedente. La classe SceltaIniziale non l'ha in quanto è la prima schermata.

- SceltaIniziale:
  - **visoneAddetto:** cambia la schermata vista dall'utente, nella schermata di login dell'addetto.
  - **visioneCittadino:** cambia la schermata vista dall'utente, nella schermata di login del cittadino.

- AccessoAddetto:
  - **accediAddetto:** cambia la schermata in "Inserisci disponibilità" o "Pannello admin" in base alle credenziali inserite.

- PannelloAdmin:
  - **inserisciAddetto:** estrapola le credenziali inserite dall'utente per comunicarle al model.

- InserisciDisponibilità:
  - **aggiornaDisponibilità:** permette di aggiornare le comboBox che mostrano le disponiblità prenotate e non.
  - **inserisci:** estrapola la disponibilità inserita dall'utente per comunicarle al model.
  - **festivo:** verifica se un giorno è festivo, quindi fine settimana o festa.
  - **pasqua:** calcola il giorno in cui ricade pasqua in un determinato anno.

- AccessoCittadino:
  - **accediUtente:** permette all'utente di accedere all'interfaccia "Scegli servizio".
  - **registraUtente:** permette all'utente di accedere all'interfaccia "Registrazione cittadino"

- RegistrazioneCittadino:
  - **registra:** metodo che estrapola le informazioni inserite dall'utente, e le comunica al model.
  - **tesseraSanitariaValida:** verifica che la tessera sanitaria inserita dall'utente sia valida.
  - **codiceFiscaleValido:** verifica che il codice fiscale sia valido.

- **soloLettere:** verifica che una stringa contiene solamente lettere o spazi.
- **ottieniConsonanti:** metodo che permette di estrapolare le 3 consonanti dal nome/cognome.
- **consonante:** verifica se una lettera è una consonante o meno.

• SceltaServizioCittadino:

- **aggiornaPrenotazioni:** aggiorna la comboBox contenente la lista delle prenotazione del cittadino.
- **avviaRicerca:** esegue la ricerca con le informazioni inserite dall'utente.
- **avviaPrenotazione:** estrapola la prenotazione inserita dall'utente e la communica al Model.
- **verificaCoerenza:** si assicura che l'utente stia rispettando i tempi previsti per la prenotazione.
- **richiestaInformazioni:** aggiorna il testo per rispondere alle domande dell'utente.

2.3 Diagramma di sequenza

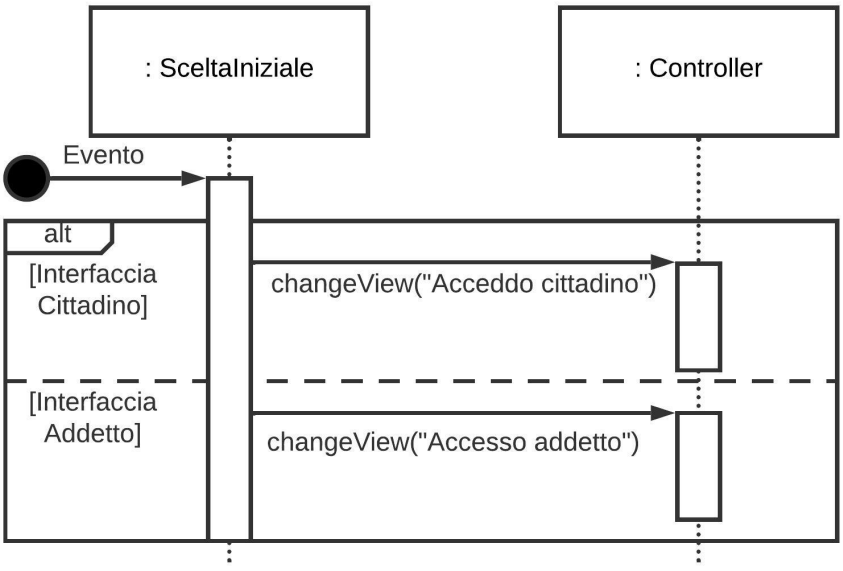


Figure 7: Scelta iniziale

Nei prossimi diagrammi, non viene rappresentata la possibilità di tornare nella schermata precedente. La sequenza di azioni è pressoché identica al diagramma

precedente. I Model nei loro metodi chiamano *"eseguiIstruzione"* e/o *"lanciaQuery"* per poter modificare/inserire/ottenere le informazioni desiderate dal database.

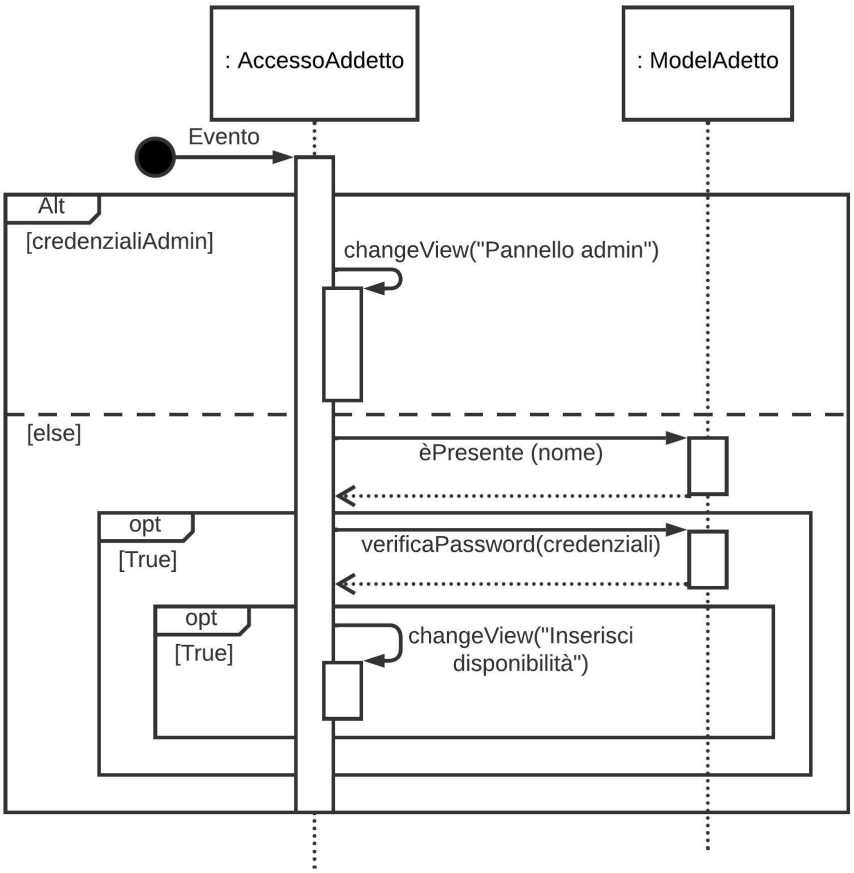


Figure 8: Accesso addetto

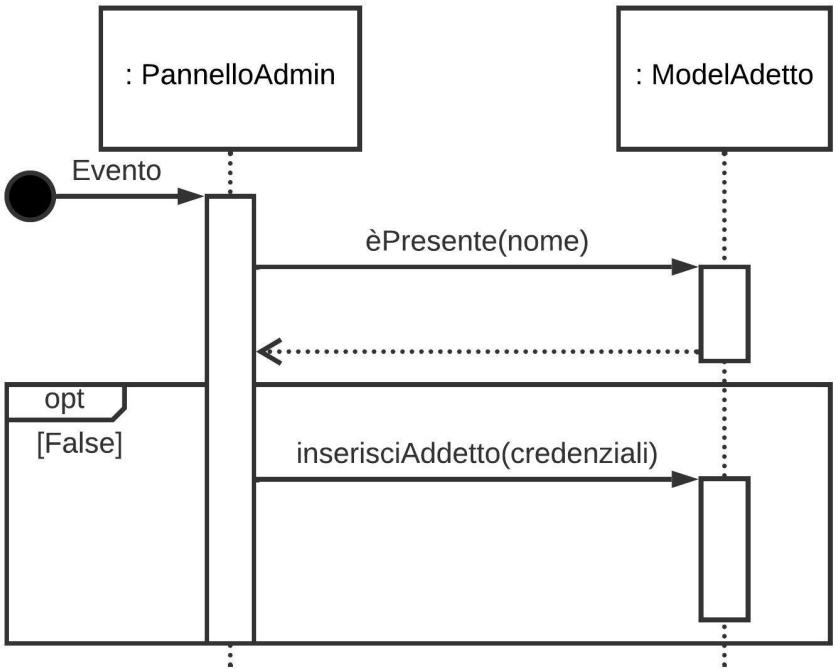


Figure 9: Pannello admin

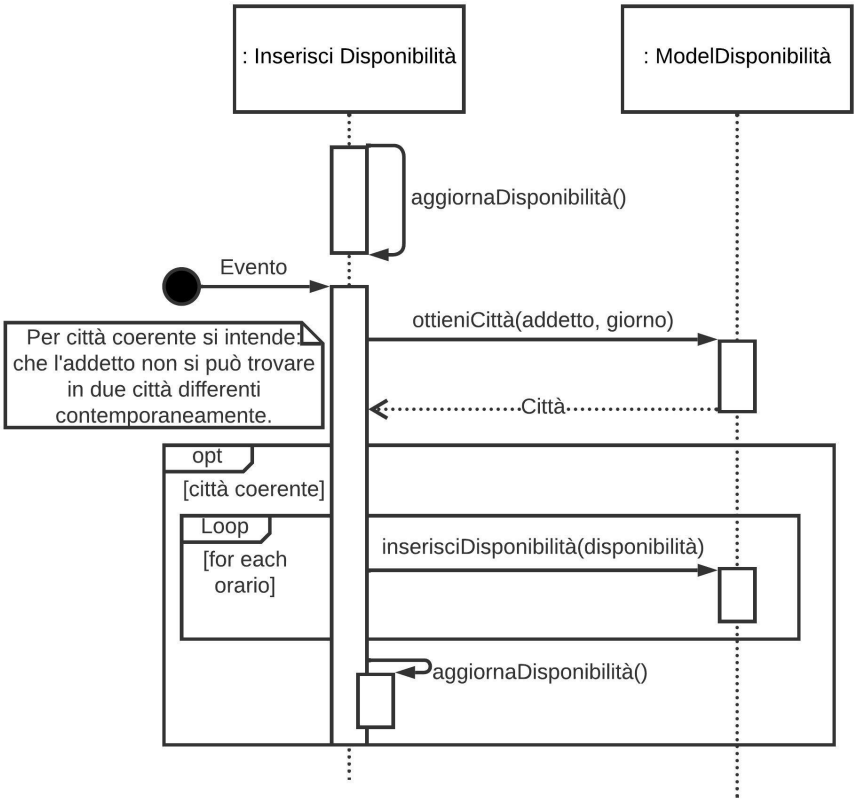


Figure 10: Inserisci disponibilità

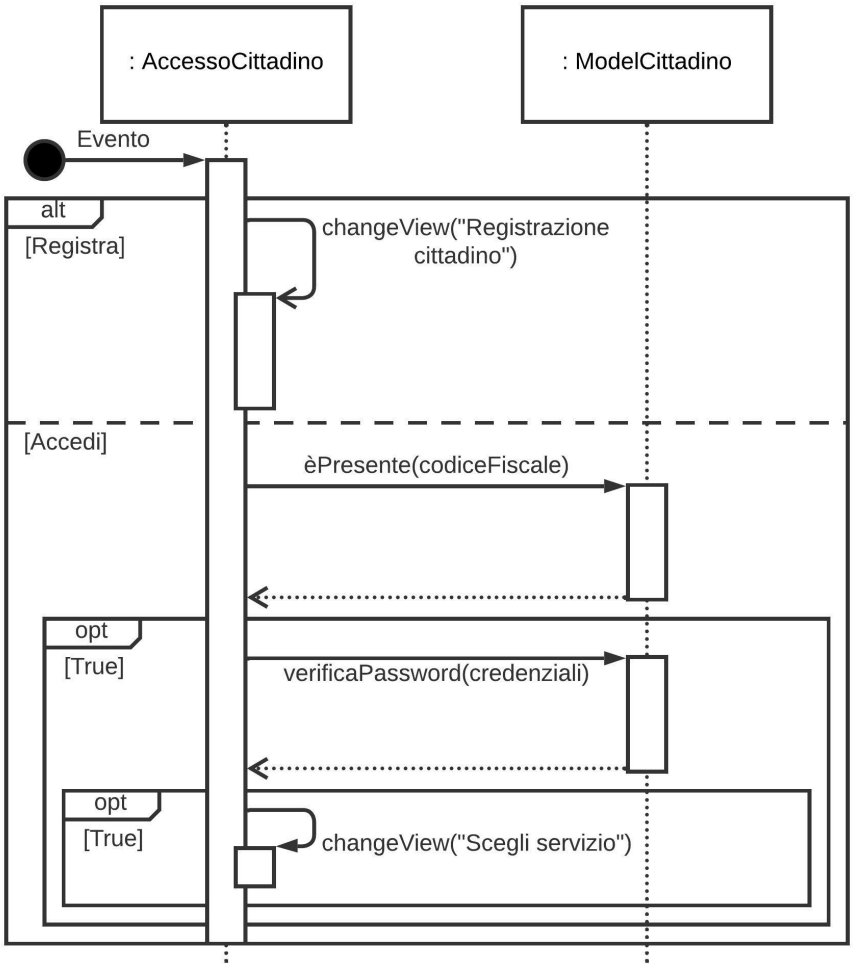


Figure 11: Accesso cittadino

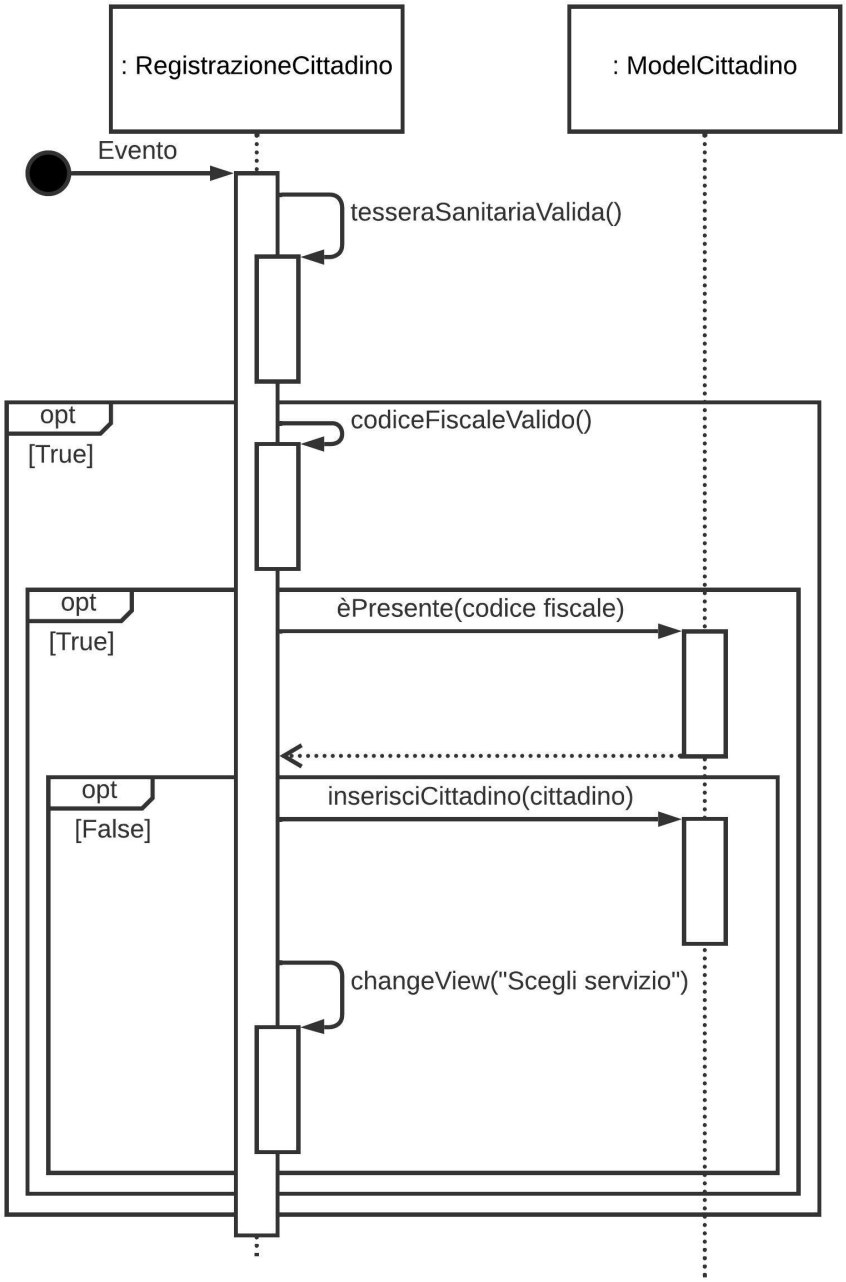


Figure 12: Registrazione cittadini

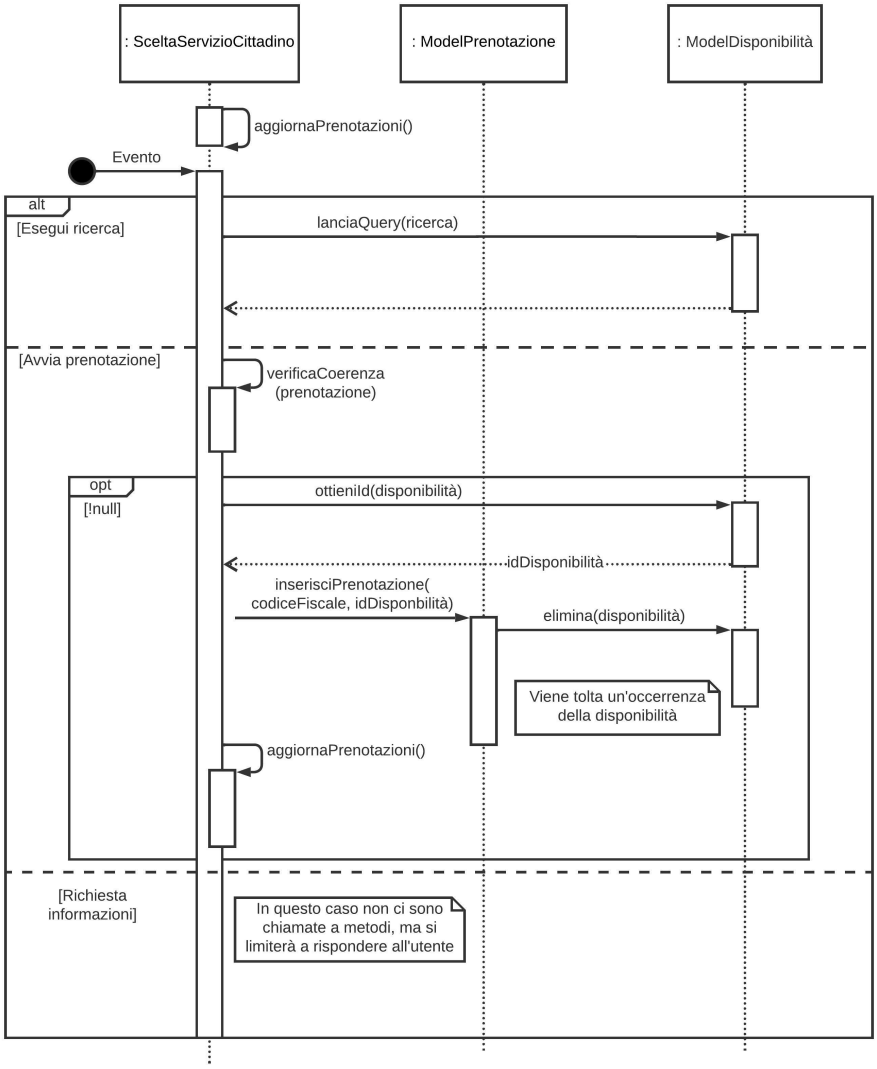


Figure 13: Scelta servizio cittadino

3 Test e validazione

In questa fase lo sviluppatore si è concentrato sul cercare comportamenti non previsti dal programma. Per far ciò il lavoro è stato suddiviso in 2 parti.

- **Analisi statica:** lo scopo principale non era quello di trovare errori logici, anche perché sarebbe stato complesso dalla solo lettura del codice, ma di

notare l'uso di notazioni differenti, zone di codice poco chiare o obsolete. Durante questo processo è stato obbligatorio fare del refactoring, seppur di importanza minimale come ad esempio, il cambiamento di un nome di variabile.

- **Test:** in questo caso ciò che veniva messo alla prova era la parte logica del programma, sottoponendo il processo a vari test semplici e non. Dai risultati dei test si passava successivamente a modificare il codice, per soddisfare eventuali test non passati.

Tutto ciò è stato ripetuto varie volte, fino a trovare quello che viene ritenuto, un buono stato del lavoro.

3.1 Unit test

I test di cui stavamo parlando fino a qualche riga fa, ovviamente, non sono stati fatti in modo manuale, sarebbe stato un lavoro noioso, ripetivo e con l'alto rischio di errore. Per questo motivo si è fatto affidamento alla libreria TestFX per assicurarsi che l'interfaccia utente rispondesse correttamente con ciò che si aspettava lo sviluppatore. Grazie a questi test si è potuta verificare la correttezza dei Controller, ma anche dei Model, dato che quest'ultimi vengono chiamati esclusivamente dal Controller. Questa fase, come precedentemente accennato, è stata ripetuta più volte, sia a progetto concluso e non. Infatti sviluppo e testing andavano avanti quasi contemporaneamente, assicurandosi che le classi e i loro metodi fossero corretti e coerenti con le specifiche iniziali, prima di continuare lo sviluppo. Una volta finito il progetto, invece, il test si è rilevato fondamentale ad ogni refactoring, fosse esso una modifica sostanziale al codice, come inserimento o eliminazione di alcuni parti, o anche minimale, come ad esempio modifiche ai nomi delle variabili.
Per sapere con maggiori dettagli i test, e di conseguenza i casi coperti durante lo sviluppo, si consiglia di controllare l'apposito package "test". I nomi dei metodi di test, con i loro appositi commenti, dovrebbero essere sufficienti per capire l'obiettivo del test.

3.2 Test utenti generici

Una volta raggiunta quella che si pensava essere la versione finale del software, si è deciso di far provare il programma a varie componenti della mia famiglia (madre, padre, 2 fratelli, fidanzata).

3.2.1 Note da sapere

Prima di passare ai risultati ottenuti dal test, bisogna sapere che tutti gli utenti, eccezione fatta per la fidanzata, sono stati costretti ad usare il software, attraverso Google Meet. Ciò significa che gli utenti non hanno potuto controllare direttamente il software, ma dicevano a voce cosa cliccare e cosa scrivere. Bisogna anche tenere in considerazione che uno dei 2 fratelli è un laureando in

Matematica, quindi potrebbe avere competenze informatiche più alte della media. In ogni test è stato chiesto di inserire delle disponibilità, quindi fare il ruolo dell'addetto, ma si è svolto anche il test lato cittadino, chiedendo all'utente di prenotare un appuntamento alla questura. I test sono stati fatti in vario ordine, quindi con alcuni utenti, si è svolto prima il ruolo del cittadino e successivamente quello dell'addetto; mentre per i restanti utenti si è svolto il contrario. Lo sviluppatore ha cercato di influenzare il meno possibile il test, fornendo le informazioni necessarie, e rispondendo ad eventuali dubbi sollecitati dagli utenti. Per ogni test sono state fornite le seguenti informazioni di base:

- **Cittadino:** vuoi fare il passaporto. Questo è il programma con cui devi farlo.

- **Addetto:** devi fare il tuo lavoro. Devi inserire la tua disponibilità in questura. Il programma con cui lo farai è questo, e le tue credenziali sono X - X.

Inoltre, all'utente è stato chiesto di sbagliare volontariamente, in caso l'avesse voluto, per verificare l'eventuale frustrazione.

3.2.2 Feedback utente

Di seguito riportiamo una tabella contenente le proposte di modifica ricevuti, la decisione presa dallo sviluppatore, con eventuale motivazione in caso di rifiuto.

Modifica	**Decisione**	**Motivazione**
Addetto in due città differenti contemporaneamente.	Accettato.	
Rendere più chiara la ricerca.	Accettato.	
L'addetto può inserire più volte la stessa disponibilità.	Rifiutata.	L'addetto potrebbe decidere che in uno slot può riuscire a soddisfare più cittadini.
L'addetto può lavorare nel fine settimana.	Accettato.	
La ricerca non rimane salvata.	Accettato.	
Inserire la possibilità di ricerca in un range di date. Ad esempio dal 05/08 al 20/08.	Rifiutata.	L'interfaccia della ricerca è una delle sezioni più complesse per l'utente da capire, aggiungere ulteriori elementi, creerebbe una maggiore confusione. L'implementazione di questa feature, mantenendo un'interfaccia usabile da un nuovo utente, richiederebbe del tempo che non c'è.
La password inseribile dall'utente non rispetta i criteri di sicurezza.	Rifiutata.	Il programma è un prototipo, per cui si è deciso di rendere più facile la navigazione agli utenti, permettendo password facili da ricordare come potrebbe essere "password" o "pass". Ovviamente in caso di un rilascio su larga scala del programma, la prima modifica sarebbe proprio questa.

Table 1: Proposte degli utenti

Alla fine del test, è stato chiesto a tutti gli utenti se hanno ritrovato particolari difficoltà, o se il programma fosse semplice e intuitivo. Tutti gli utenti hanno risposto alla domanda, affermando che fosse facile giungere all'obbiettivo.

3.2.3 Modifiche dello sviluppatore

Grazie a questi test, lo sviluppatore si è potuto immedesimare nell'utente finale, scoprendo delle zone del programma che potevano essere utilizzate incorrettamente. Di seguito, riportiamo una piccola lista contenente gli aggiustamenti che, probabilmente, non sarebbero nati senza i test degli utenti.

- Uso di parole differenti nell'interfaccia grafica, cosi da risolvere potenziali dubbi di un nuovo utente.

- Sistema la possibilità di prenotare un rilascio, un ritiro un mese dopo, e un rilascio nel mezzo. Infatti prima dei test degli utenti, a causa di un errore era possibile:

  1. Prenotare un rilascio per il giorno 10/07.
  2. Prenotare un ritiro per il giorno 20/08.
  3. Prenotare un rilascio per il giorno 15/07.

  Ovviamente, ciò non doveva essere ammesso, in quanto si stava prenotando un rilascio senza aver effettuato tutti i ritiri.

- Aggiunti controlli sui testi scrivibili dall'utente, per assicurarsi che non superassero i 255 caratteri.