

# Relazione progetto

A stylized, cursive logo that reads "Sedibus". The letters are fluid and interconnected, with a prominent 'S' at the beginning and a 'b' at the end.

Nicola Costa

Rostyslav Dovganyuk

Vladislav Lapi Miruk

25 aprile 2021

# Indice

<b>1 Analisi.....</b>	<b>3</b>
<b>1.1    Requisiti .....</b>	<b>3</b>
<b>1.2    Analisi e modello del dominio .....</b>	<b>4</b>
<b>2 Design.....</b>	<b>5</b>
<b>2.1    Architettura .....</b>	<b>5</b>
<b>2.2    Design dettagliato.....</b>	<b>6</b>
<b>3 Sviluppo .....</b>	<b>11</b>
<b>3.1    Testing .....</b>	<b>11</b>
<b>3.2    Metodologia di Lavoro.....</b>	<b>11</b>
<b>3.3    Note di Sviluppo .....</b>	<b>12</b>
<b>4 Commenti finali.....</b>	<b>13</b>
<b>4.1    Autovalutazione e lavori futuri .....</b>	<b>13</b>
<b>4.2    Difficoltà incontrate e commenti per i docenti .....</b>	<b>14</b>
<b>A Guida Utente.....</b>	<b>15</b>

# Capitolo 1

## Analisi

Il software è un gestionale completo per un ristorante, che oltre a offrire la possibilità di prenotare, permette anche di scegliere il tavolo, in quanto il progetto principale si contorna proprio al concetto di una visione interattiva e schematizzata del ristorante.

L'obiettivo è di rendere il programma funzionale sia per il personale del ristorante, che per i clienti che non riescono a chiamare il ristorante stesso per prenotarsi.



FIGURA 1.0 - L'idea da cui nasce il progetto

## 1.1 Requisiti

### Requisiti funzionali

- All'avvio l'applicazione offre la possibilità di scegliere il tipo di utente con cui accedere.
- All'apertura della piantina i tavoli vengono direttamente visualizzati del colore corrispondente al loro stato, in base alla data corrente e al periodo selezionato.
- Se l'utente che sta visualizzando la piantina è un Admin, ha la possibilità di visualizzare le informazioni dei tavoli prenotati e eliminare o modificare la prenotazione corrispondente al tavolo.
- Possibilità di modifica e cancellazione delle prenotazioni da parte di entrambi gli utenti.

### Requisiti non funzionali

- Grafica minimale, interattiva e molto intuibile.
- Veloce reperibilità dei dati.

## 1.2 Analisi e modello del dominio

All'avvio dell'applicazione si dovrà poter scegliere il tipo di utente con cui si vogliono eseguire le successive azioni (Admin, Utente).

Nel caso si voglia accedere come admin verranno richieste le credenziali, mentre come Utente si avranno diverse opzioni: prenotare, modificare o cancellare.

Sedibus offre una visuale dall'alto della collocazione dei tavoli nel ristorante ed in base all'utente con cui si è fatto accesso offre un diverso tipo di permesso e interazione coi tavoli. I tavoli prenotati saranno visualizzati di color rosso, mentre quelli liberi verdi.

Se l'utente è admin, sul tavolo rosso visualizzerà le informazioni relative alla prenotazione e avrà la possibilità di modificarla/eliminarla.

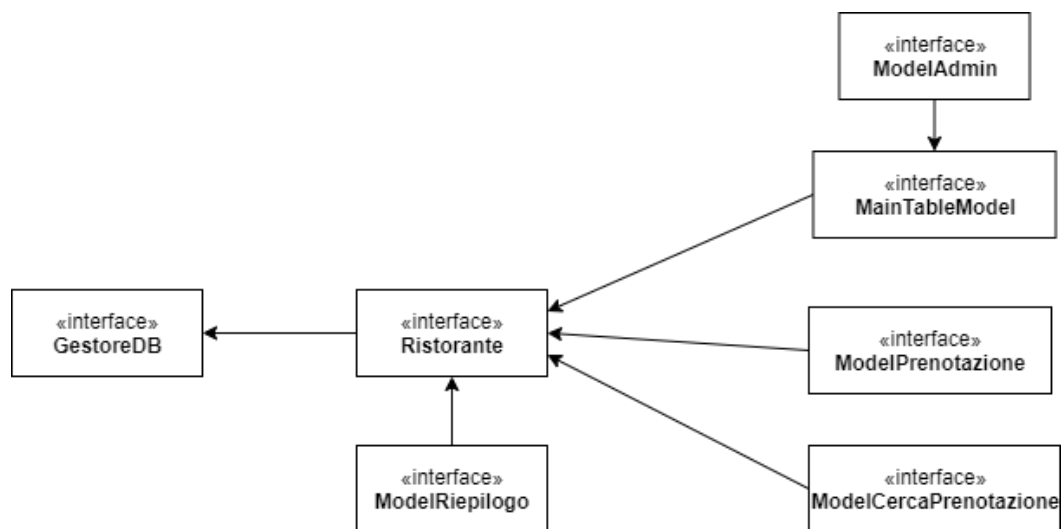


FIGURA 1.1 - Schema UML del Dominio

## Capitolo 2

### Design

#### 2.1 Architettura

Il modello segue un pattern “multi” MVC, dove ogni macro-componente dell’applicazione ha una propria View, Controller e Model, infatti il software ricorda molto il modello delle pagine Web.

Il compito del Model Principale è quello di interfacciarsi col gestore del database, mentre i Model “satellite” si occupano della modellazione dei dati in base alle informazioni passate dai Controller.

Un esempio è la mappa dei tavoli dove l’utente sceglie il tavolo da prenotare: il Model elabora gli elementi necessari in base alle variabili passate dal controller, che imposterà visualizzazione e i colori dei tavoli, e allo stesso tempo si occupa di aprire le View necessarie, in base alle scelte dell’utente.

Il motivo principale per il quale si è optato per tale pattern sono le innumerevoli combinazioni per poter estrarre le informazioni necessarie avendo diversi dati di input, e per non andare ad appesantire un unico Model.

Questa scelta rende difficile sostituire la corrente libreria grafica a causa delle sue peculiari notazioni necessarie per prelevare/settare i vari componenti grafici.

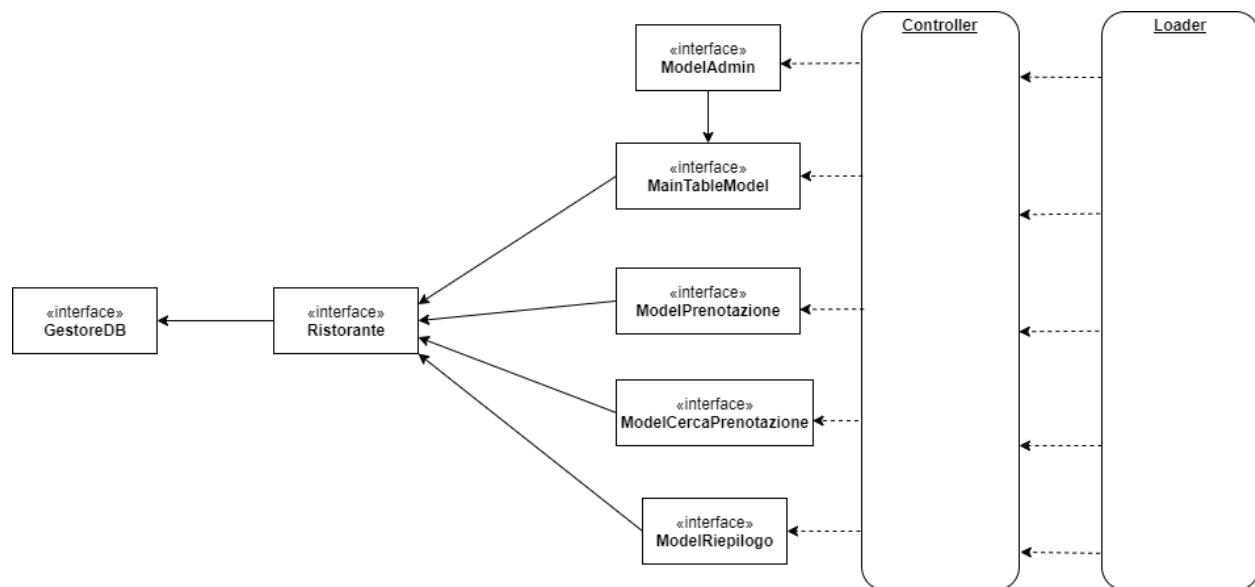


FIGURA 2.0 – Schema UML dell’architettura

## 2.2 Design dettagliato

### SELEZIONE UTENTE

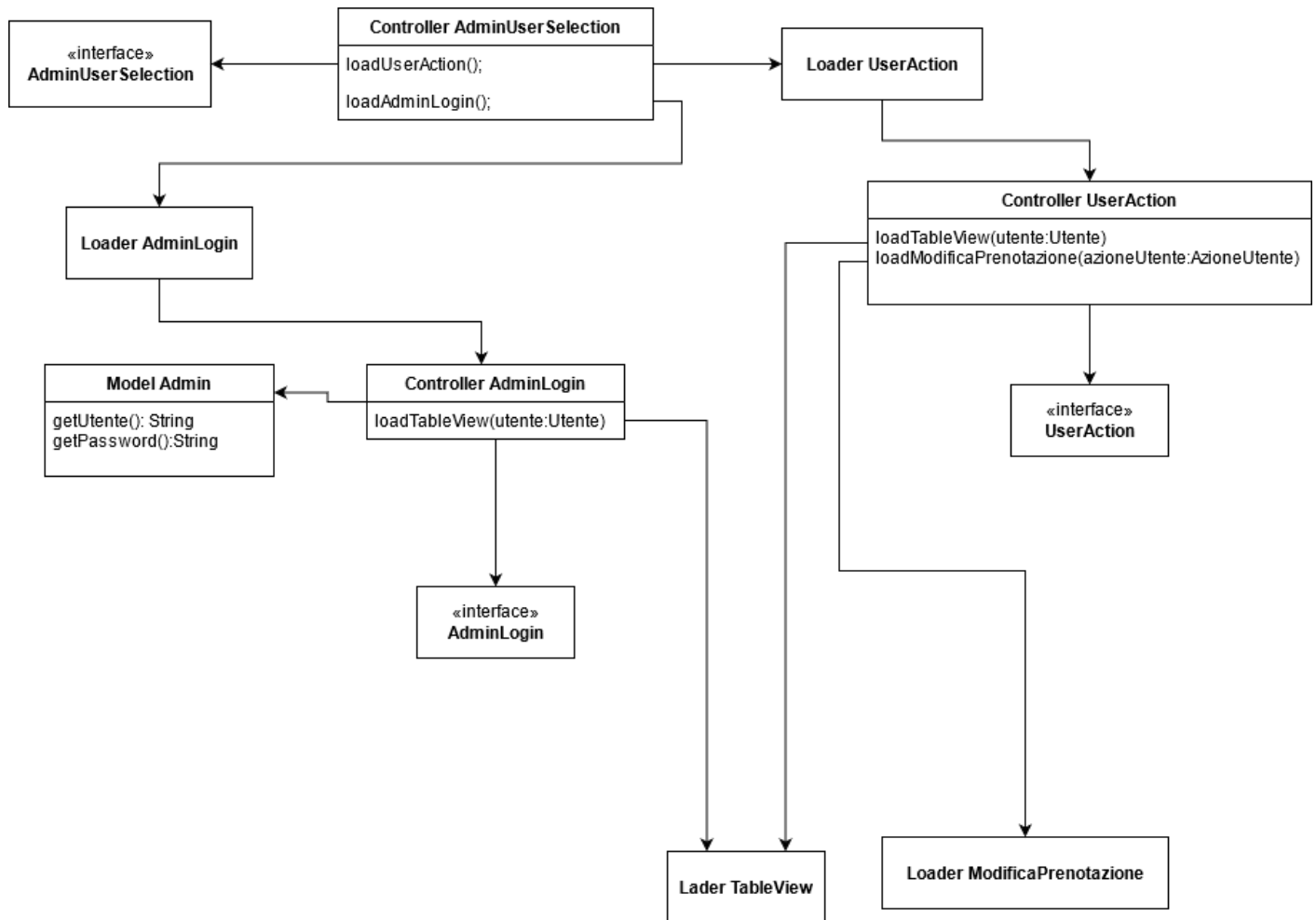


FIGURA 2.1 – Schema UML della Selezione Utente

La parte di selezione utente segue il pattern MVC solamente nella parte AdminLogin mentre le altre parti spesso manca il Model in quanto si occupano solamente di caricare una nuova finestra e non necessitano di una parte logica.

La model Admin è la parte che si occupa di implementare la libreria Gson per la lettura del file json contenente i dati dell'amministratore.

Per determinare se un utente sia admin oppure no abbiamo optato per un Enum Utente e anche per determinare l'azione che un utente desidera svolgere è stato deciso di usare un Enum AzioneUtente.

## GESTIONE DATABASE

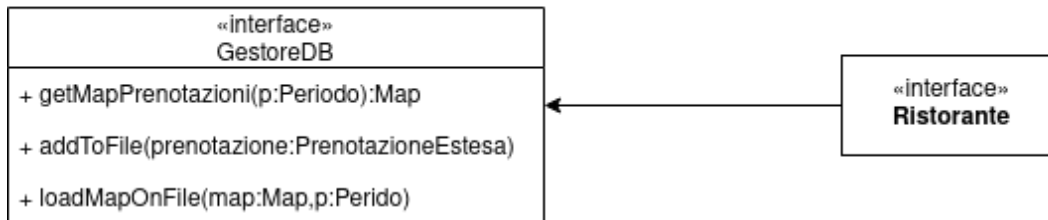


FIGURA 2.2 – Schema UML della Gestione DataBase

Per il salvataggio delle informazioni su file si è optato al formato .json, in quanto semplice da leggere e molto funzionale per questo tipo di applicazione, per tale compito è stata usata la libreria esterna “*gson*”.

Per rispettare le nostre idee iniziali di visualizzazione, i dati vengono suddivisi in due file, che corrispondono ai due periodi (Pranzo e Cena) di lavoro del ristorante, e viene utilizzata la struttura map che, come chiave, ha la data e i suoi valori corrispondenti sono le prenotazioni.

Il funzionamento del database è alquanto articolato siccome non sapevo come poter prelevare solo le informazioni necessarie, oppure salvare una prenotazione senza eliminare tutto il contenuto del file, perciò prima di salvare una prenotazione si è optato a estrapolare in formato mappa i dati dal file .json inerente al periodo della prenotazione, e aggiungere la prenotazione alla lista, con relativi controlli sulle chiavi.

## VISUALIZZAZIONE PIANTINA

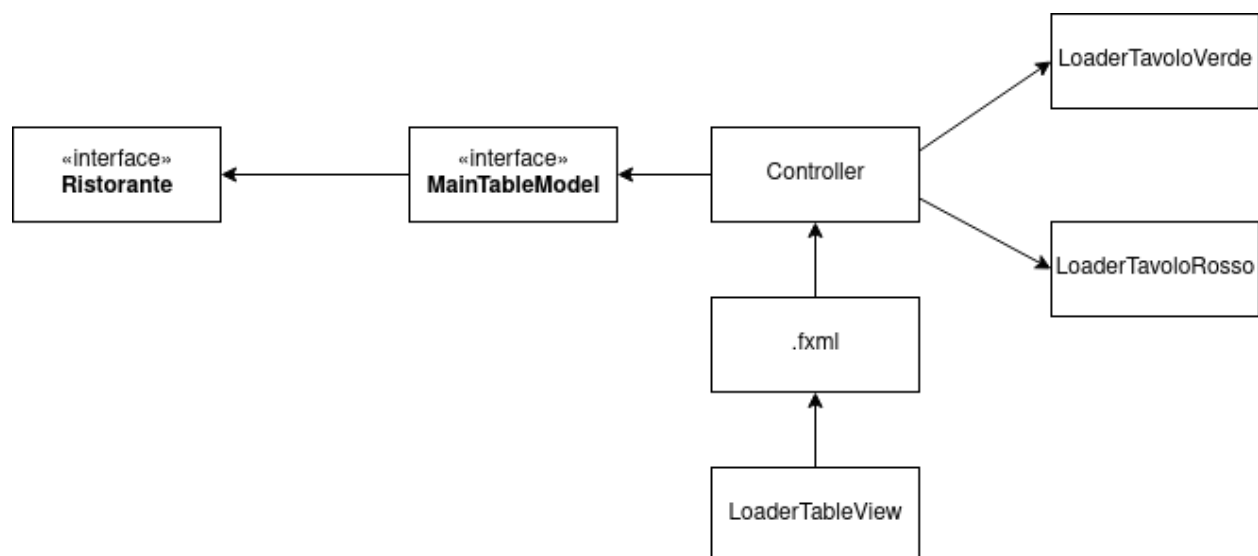


FIGURA 2.3 – Schema UML della Visualizzazione Piantina

In questa sezione viene usato il pattern MVC, o più propriamente MMVC, il primo Model che si occupa di estrapolare i dati dal database, e il secondo Model che li modella in base ai dati di input passati dal controller, infatti si è pensato di usare tale “architettura” per non appesantire una unica model e tenere separati i lavori di Model e Controller.

L’idea principale, sulla quale si focalizza il progetto, è la visualizzazione dei tavoli in base al loro stato, verde se libero e rosso se occupato.

La Main Table Model è costruita su misura per avere una modellazione dati, che andrebbe soltanto ad appesantire un solo ipotetico Model.

Il controller si occupa di gestire i colori dei tavoli, gli handler sul menù principale e gli handler sui tavoli.

Infatti tutte le volte che viene cambiata la data o il periodo il controller, con il supporto del Model, si occupa di rielaborare i dati e ricolorare i tavoli in base ai loro stati, applicando loro il foglio di stile (.css) giusto.

Ovviamente in base all’utente, la visualizzazione non cambia, ma cambiano le funzionalità, una importante è la possibilità per l’admin di visualizzare le informazioni di un tavolo rosso, solo cliccandoci sopra, funzionalità ovviamente vietata a un utente normale, un’altra funzionalità in più è anche quella di poter eliminare o modificare la prenotazione, senza andare a ripescare il codice prenotazione e cognome.

Alla pressione del tavolo verde viene aperta una piccola finestra, che in base al bottone cliccato dall’utente riporta alla visualizzazione della piantina del ristorante oppure a una nuova pagina, inerente completamente alla prenotazione.

Il tavolo rosso, invece, si attiva solo quando l’utente è admin, aprendo così una pagina che mostra le informazioni della prenotazione, e offre la possibilità di eliminarla o modificarla.

## **CREAZIONE PRENOTAZIONE**

La LoaderPrenotazione si occupa di caricare il file ScenePrenotazione.fxml che contiene gli elementi grafici e di istanziare il ControllerPrenotazione. A quest’ultimo vengono passati nel suo costruttore i parametri necessari alla corretta presentazione all’utente della View: gli Enum Utente e AzioneUtente che fanno da flag. Per quanto riguarda la parte di creazione di una prenotazione si è deciso di creare una finestra con diversi campi completabili dall’utente. PilotaPosti è la classe che si occupa di gestire i posti che l’utente vuole prenotare. Una volta cliccato il pulsante di conferma vengono fatti pochi semplici controlli sui campi col metodo controllaDatiCliente e, se corretti, si aggiunge la prenotazione al database grazie al metodo di aggiunta di Ristorante, incapsulato nel metodo aggiungiPrenotazione di ModelPrenotazione. Per la generazione del codice prenotazione si è deciso di produrlo con il seguente pattern: 1 lettera casuale + 4 cifre che iniziano da 0000; il primo codice sarà dunque lettera0000, il secondo lettera0001, ...; questo garantisce l’univocità dei codici. Ristorante, GeneratoreCodice e PilotaPosti sono istanziati internamente al ModelPrenotazione per racchiudere in un’unica classe la parte di logica della creazione di una prenotazione.



## MODIFICA - CANCELLAZIONE

La modifica di una prenotazione è legata alla creazione, cioè si utilizza la stessa View, Model e Controller. Dopotutto, si è deciso di attuarla prelevando i dati in ingresso ad un secondo apposito costruttore della LoaderPrenotazione e settare i testi dei componenti grafici con essi. Il cliente può inoltre modificare la data e il turno pranzo/cena grazie al metodo cercaTavolo che restituisce il primo tavolo libero che soddisfa i requisiti. Una volta cliccato il pulsante di conferma il metodo modificaPrenotazione cancella la vecchia e aggiunge la nuova, sempre con lo stesso codice prenotazione, grazie ai metodi di Ristorante. Per modificare o cancellare una prenotazione occorre prima cercarla: l'MVC CercaPrenotazione permette di ricercare con cercaDati del Model una prenotazione presente nel database inserendo il codice, il cognome del cliente e il turno che si era scelto per trovarla e, se il flag (passato al LoaderCercaPrenotazione) è di MODIFICA, allora il ControllerCercaPrenotazione porta alla LoaderPrenotazione, mentre se è CANCELLA allora viene richiamato il metodo eliminaPrenotazione del ModelCercaPrenotazione. L'eliminazione viene effettuata molto semplicemente mostrando all'utente la ViewAlert per confermarla.

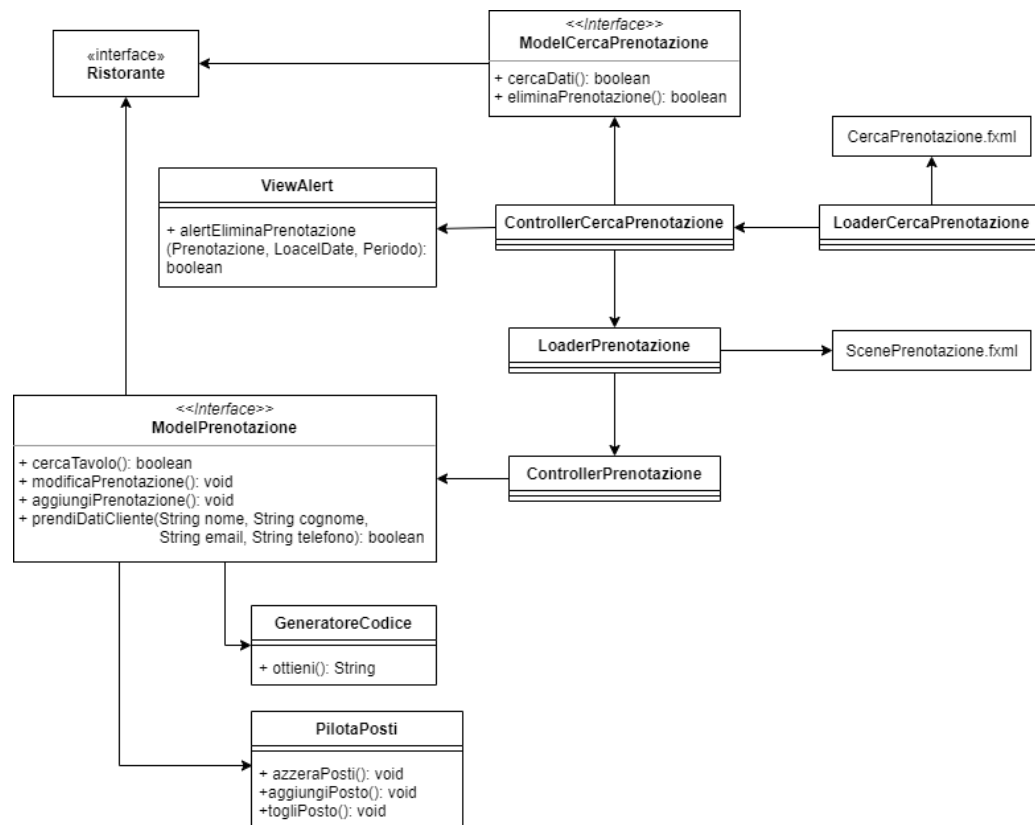


FIGURA 2.4 – Schema UML della Modifica e Cancellazione

## **RIEPILOGO**

Il riepilogo di una prenotazione si occupa di mostrare all'utente il riassunto dei dati che ha appena inserito/modificato ed anch'esso segue il pattern MVC: la View è composta dal Loader e dal file Riepilogo.fxml (il primo carica il secondo), il Controller è istanziato nel Loader per consentire il passaggio di parametri fondamentali da View a Controller. Il Model fa uso di Ristorante per cercare nel database la prenotazione e poi spacchetta i vari campi per inserirli, tramite getters, nei componenti della View, grazie al Controller che li mette in comunicazione.

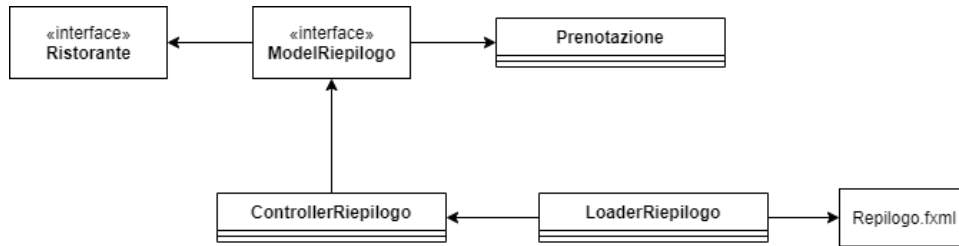


FIGURA 2.5 – Schema UML del Riepilogo

## Capitolo 3

### Sviluppo

#### 3.1 Testing

Il testing effettuato non è stato di tipo automatizzato in quanto mirato principalmente alla visione grafica dell'insieme, come ad esempio la colorazione dei tavoli in caso di tavolo occupato, o anche la visualizzazione delle informazioni inerenti al tavolo.

#### 3.2 Metodologia di Lavoro

L'analisi del problema è stata svolta collettivamente prima dell'inizio della fase di sviluppo, concentrandoci molto sulla divisione dei compiti e sul funzionamento dell'applicazione nel tutt'uno.

Il lavoro è stato suddiviso nel seguente modo:

- Vladislav Lapi Miruk: la selezione del tipo di utente viene gestita dalla classe AdminUserSelection che si occuperà di caricare la schermata per proseguire come amministratore oppure utente.  
Nel caso si voglia proseguire come utente si procede verso la classe UserAction che passa un Enum Utente impostato su utente alla classe loader TableView nel caso si voglia creare una nuova prenotazione, oppure viene passato un Enum AzioneUtente alla classe LoaderModificaPrenotazione con la tipologia di azione scelta dall'utente nel caso si voglia modificare o cancellare un ordine.  
Nel caso si voglia usare il programma come amministratore viene caricata la classe AdminLogin che userà i metodi presenti nel Model Admin, che vanno a leggere i dati presenti nel file json, per confrontare i dati inseriti dall'utente con quelli presenti sul file, in caso i dati combacino viene impostato l'Enum Utente su admin e passato alla classe loader TableView, in caso contrario viene impostata la label con un messaggio di errore finché l'utente non inserirà i dati corretti.  
Ogni schermata caricata è stata realizzata con un Responsive Design in che il programma venga visualizzato correttamente su ogni tipo di risoluzione dello schermo anche se non è stato possibile eseguire dei test su monitor con risoluzione superiore al Full HD.  
Per garantire una corretta integrazione tra le parti il LoaderCercaPrenotazione è stato creato in collaborazione con Nicola
- Rostyslav Dovganyuk: creazione della “mappa” di collegamento tra le interfacce grafiche con il supplemento di programmi Adobe  
Visualizzazione della mappa tavoli dall'alto, compresa la colorazione dei “suddetti” tavoli, e le eventuali componenti in caso di pressione su tavoli rossi o verdi.  
Estrapolazione dei dati dal DataBase, il relativo Model principale.
- Nicola Costa: creazione, modifica e riepilogo di una prenotazione. Gestione delle eccezioni.
- L'eliminazione è stata gestita da Rostyslav e curata da Costa e Vladislav.

Seppur il lavoro è stato suddiviso in maniera netta, le eventuali modifiche o idee sono sempre state comunicate e prese in considerazioni collettivamente, con meeting frequenti per controllare l'andamento dei lavori.

È stato usato “Git” come DVCS per rendere le cose semplici. Non avendo grandi esperienze nel campo, è stato usato un unico branch principale, in modo da tenersi sempre a pari, e avere il lavoro degli altri componenti del gruppo sempre aggiornato nel proprio progetto.

### 3.3 Note di Sviluppo

Vladislav

- JavaFX + FXML
- Gson
- InputStream, Reader per leggere dai file

La libreria Gson permette di scrivere e leggere dati da un file json

Rostyslav Dovganyuk:

- JavaFX + FXML;
- Gson;
- Lambda expression e stream ove possibile per migliorare la qualità del codice;
- Optional quando necessitavo di oggetti non null;
- InputStream, reader e writer per la lettura/scrittura dati da file;
- Type Token per deserializzare correttamente da Json al tipo di dato necessario, in questo caso alla Mappa.  
(grazie al forum studenti)

Nicola Costa:

- JavaFX + FXML;
- Lambda Expression;
- Stream;
- Optional
- BufferedWriter + FileWriter
- BufferedReader + FileReader

## Capitolo 4

### Commenti finali

#### 4.1 Autovalutazione e lavori futuri

##### Vladislav Lapi Miruk

Il prodotto finale ha come punto di forza quello di essere semplice e intuitivo per l'uso da parte di un utente, mentre la sua debolezza sta nel fatto che componenti come i dati per l'accesso da parte dell'amministratore non siano modificabili all'interno del programma da parte dell'utente e richiedono una modifica del codice o dei file json, Inoltre è possibile avere solamente un amministratore alla volta.

Essendo una persona esterna al gruppo di lavoro il mio ruolo è stato quello di dare un'opinione e delle idee più imparziali cercando di lavorare sempre in sintonia con gli altri membri del gruppo.

Sarebbe interessante portare avanti il progetto eliminando i suoi punti deboli, e facendo in modo che l'applicazione possa essere usata anche in remoto attraverso il web.

##### Nicola Costa

Ho trovato lo sviluppo di questo software un'esperienza formativa, in quanto è la prima volta che ho lavorato ad un progetto di medie dimensioni come questo, insieme ad un team. Dal punto di vista relazionale mi sono trovato molto bene, ho sempre dato la mia opinione sulle scelte da fare e sono stato sempre ascoltato, tuttavia quello che sento da criticarmi è l'organizzazione e l'analisi del problema: è stata la mia prima volta (per un progetto di queste dimensioni) e sono conscio di aver fatto una scarsa analisi del dominio e di tutto quello che di conseguenza viene dopo; JavaFX secondo me è stato molto complicato inizialmente da capire e utilizzare e probabilmente è stato quello che ha stravolto il mio pensiero originale di sviluppo del progetto ma è stato necessario per rendere il tutto più moderno e giustamente impegnativo.

L'esperienza darà i suoi frutti, secondo me, il prossimo o prossimi anni dato che mi ha reso sicuramente più preparato.

Il software a mio avviso è molto semplice dal punto di vista View, il che lo rende molto user-friendly e utilizzabile realmente come applicativo web viste le "analogie".

Non continuerò i suoi ipotetici sviluppi futuri.

##### Rostyslav Dovganyuk

Posso dire di essere abbastanza soddisfatto del nostro lavoro in generale, visto che il funzionamento corrisponde all'idea iniziale, ma anche per l'atmosfera lavorativa che si è creata durante questo progetto.

Come gruppo di lavoro mi sono trovato molto bene, seppur alcuni sono colleghi non li conoscevo proprio, e sono contento della professionalità che c'è stata tra di noi, le idee non sono mai state prese di testa propria, ma ci si è sempre consultati e aiutati a vicenda.

Ovviamente i problemi durante il percorso non sono mancati, ma tutto sommato abbiamo reagito bene, e abbiamo svolto egregiamente il nostro lavoro.

Per quanto riguarda una futura rielaborazione del progetto preferirei renderlo una web app con la possibilità di salvare i dati delle prenotazioni in un vero e proprio database relazionale, modificare la grafica e rielaborare un backend più efficiente, e soprattutto la possibilità per il ristorante di crearsi la loro mappa e caricarla in base alle loro necessità, e poterlo commercializzare, in quanto potrebbe davvero essere comodo sia per i ristoranti sia per i clienti, anche per dare una spinta in più ai piccoli ristoranti di quartiere.

## **4.2 Difficoltà incontrate e commenti per i docenti**

Vladislav Lapi Miruk: Essendo una persona esterna al gruppo all'inizio non sapevo se sarei riuscito a integrarmi

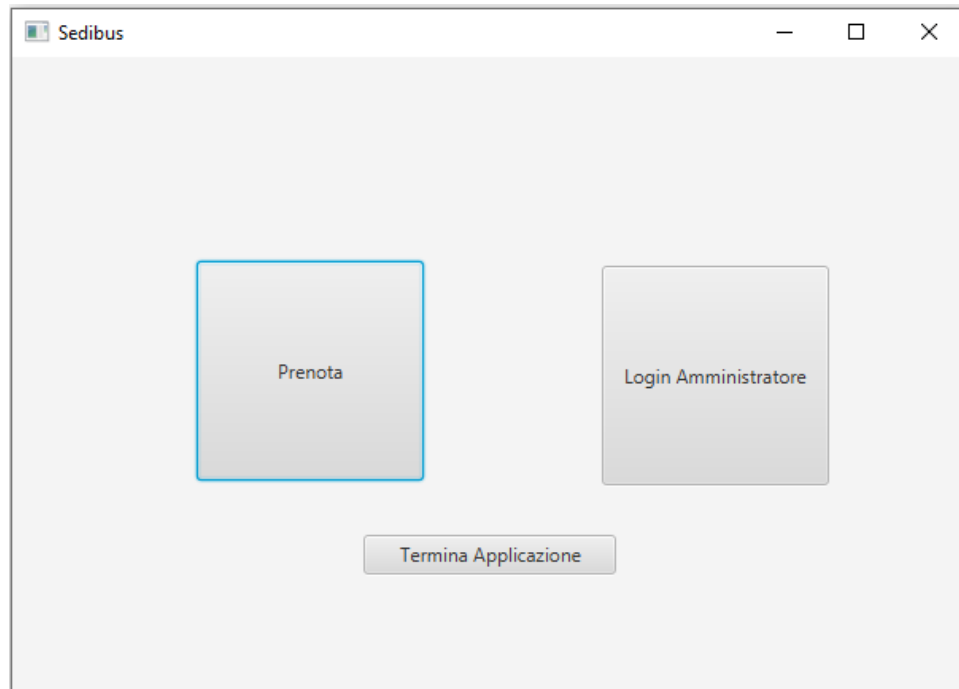
Rostyslav Dovganyuk: - creazione del Jar, il funzionamento di javafx

Nicola Costa: funzionamento JavaFX e analisi e modello del dominio.

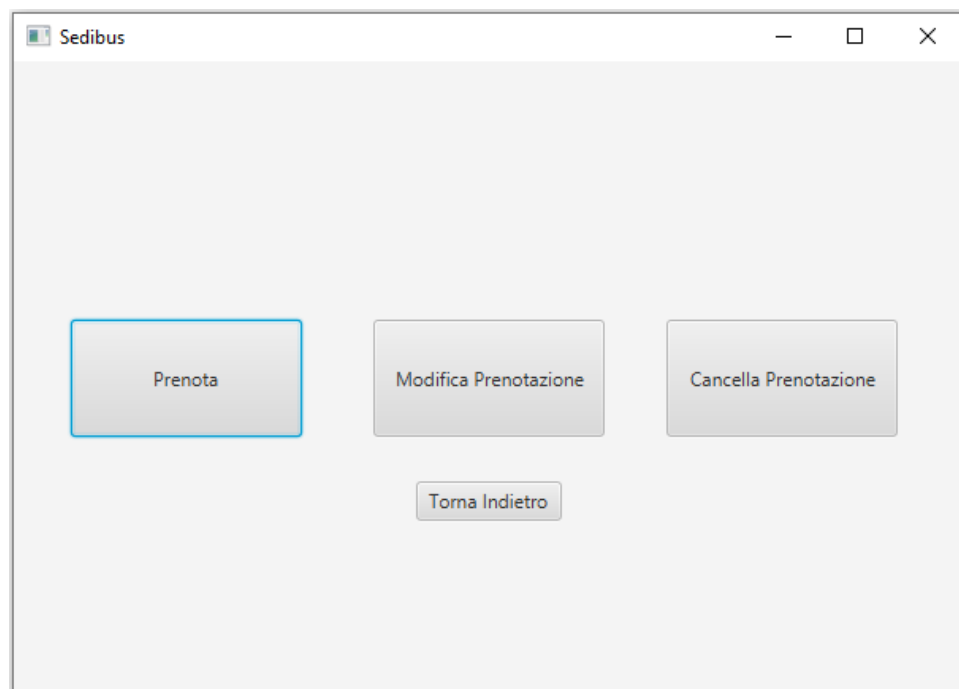
Siamo tutti e tre d'accordo che l'abbandono del nostro quarto componente non ha influito troppo sulla creazione del software ma ci ha portato via tempo per la stesura di questa relazione.

# Appendice A

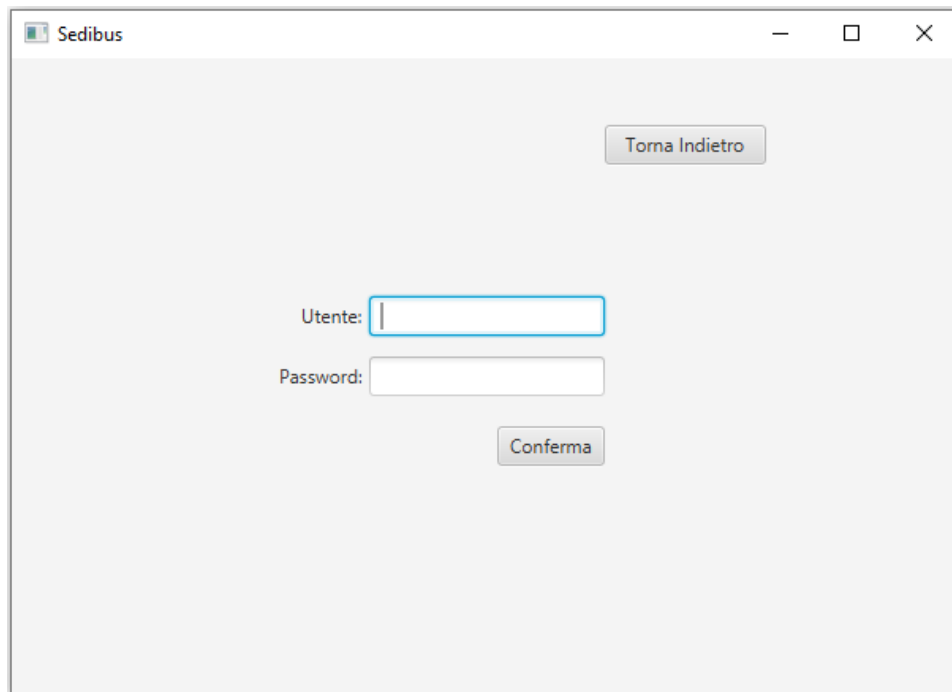
## Guida Utente



Nella schermata iniziale l'utente dovrà scegliere se vuole usare il programma come un utente normale, cliccando su Prenota, oppure come amministratore cliccando sul pulsante Login Amministratore



Nel caso si decida di accedere come utente, viene aperta una schermata dove l'utente dovrà decidere quale azione vuole eseguire



The screenshot shows a window titled "Sedibus" with a light gray background. At the top right, there are standard window controls (minimize, maximize, close). Below these, there is a button labeled "Torna Indietro". In the center, there are two input fields: "Utente:" followed by a text box, and "Password:" followed by a text box. Below the password field is a button labeled "Conferma".

Nel caso invece si decida di accedere come amministratore viene aperta una schermata dove vengono richieste le credenziali per l'accesso che di base sono impostate su **Utente:** utente **Password:** password



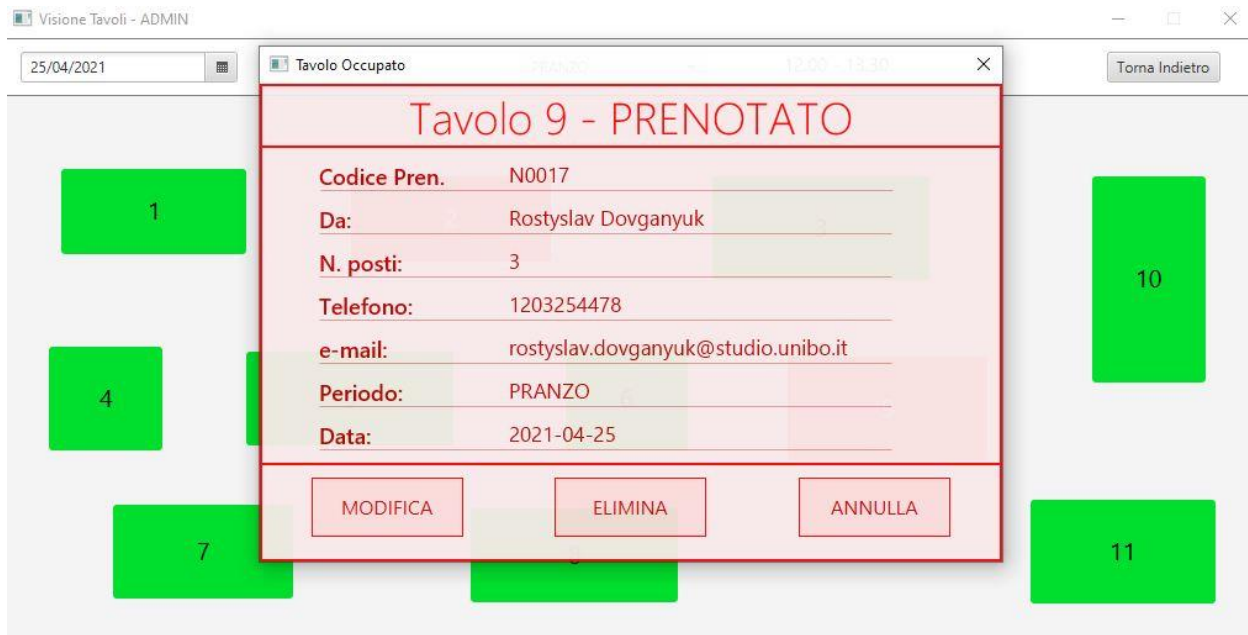
The screenshot shows a window titled "Visione Tavoli - ADMIN". At the top, there is a date input field showing "25/04/2021" with a calendar icon. Next to it is a dropdown menu showing "PRANZO". To the right of the dropdown is the time range "12.00 - 13.30". At the top right, there is a button labeled "Torna Indietro". Below the header, the word "ADMIN" is centered. The main area displays 11 numbered table icons arranged in a grid. Most icons are green, but icons 2 and 9 are red. The icons are numbered 1 through 11.

La mappa tavoli si presenta in questo modo, e permette la scelta della data e del periodo nel menù superiore, i colori dei tavoli visualizzati corrispondono alla data e al periodo inseriti nel menù.



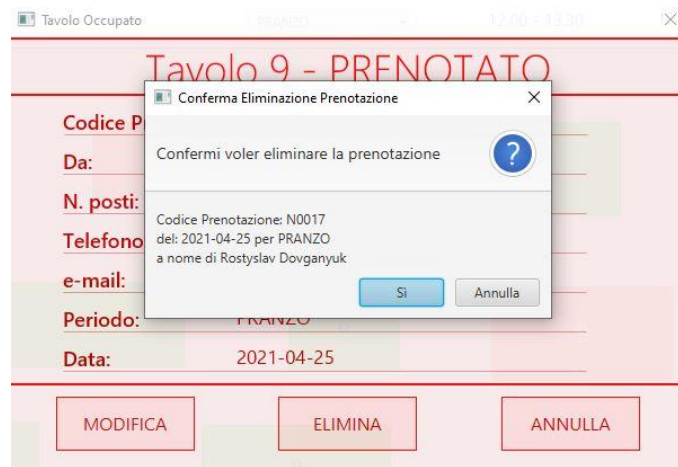


Il click sul tavolo verde, indipendentemente dal tipo di utente, appare in questo modo, mentre in caso di click su tavolo rosso, solo se l'utente è ADMIN:



Offre così la possibilità all'admin di eliminare la prenotazione di quel tavolo, o eliminarla.

Nel caso di eliminazione viene visualizzato un messaggio che richiede la conferma di tale operazione:



Mentre per la modifica viene aperta la finestra di Prenotazione con i dati della prenotazione già precompilati.

The screenshot shows a reservation form with the following fields and values: 'Nome' (Mario), 'Cognome' (Rossi), 'E-mail' (mariorossi1@gmail.com), 'Numero telefonico' (+39 3809030455), 'Data' (25/04/2021), 'Turno' (PRANZO), and 'Numero posti' (3). At the bottom of the form are three buttons: 'CONFERMA', 'RESETTA', and 'ANNULLA'.

Schermata che viene mostrata quando si vuole creare una prenotazione, è la stessa anche nel caso della modifica ma con i campi Data e Turno modificabili.



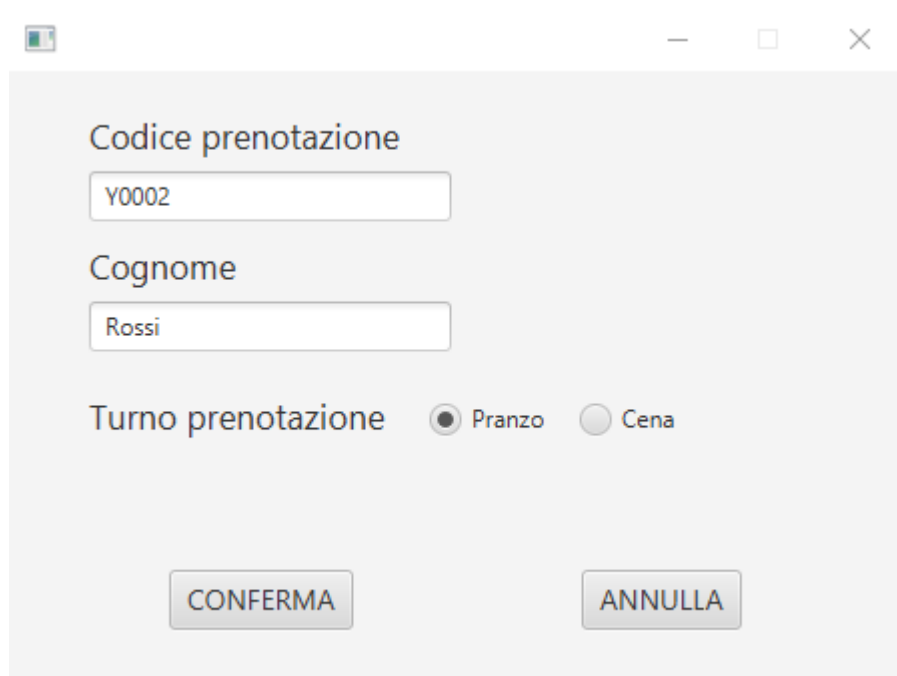
Prenotazione creata

**Riepilogo:**

Nome: Mario  
Cognome: Rossi  
Email: mariorossi1@gmail.com  
Numero telefonico: 3809030455  
Data: 2021-04-25  
Periodo: pranzo  
Posti: 3  
Tavolo selezionato: 5  
**Codice prenotazione: Y0002**

FINE

Riepilogo della prenotazione appena creata con il codice prenotazione univoco.



Codice prenotazione

Y0002

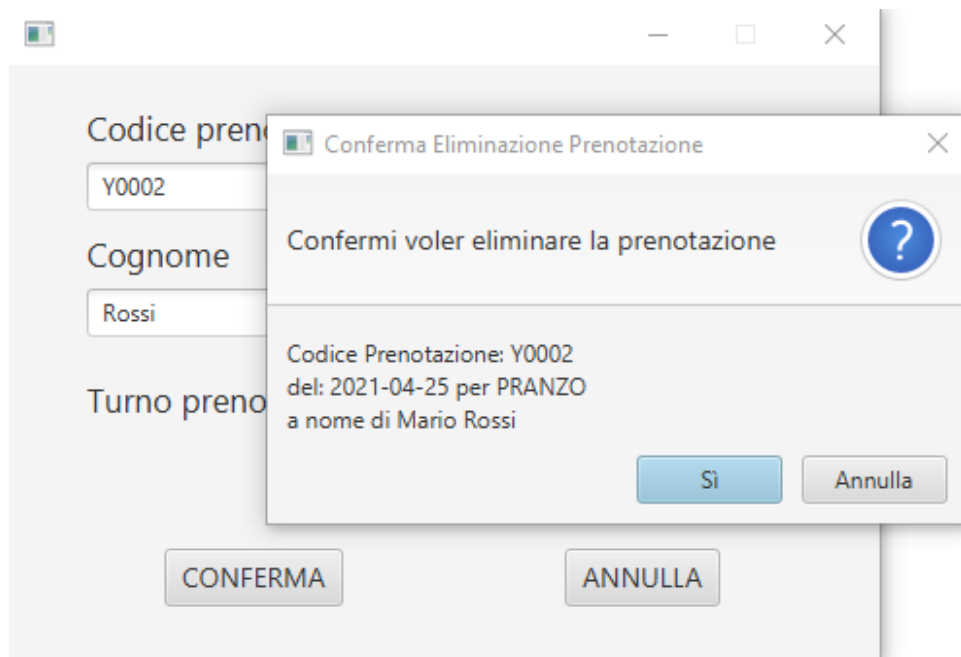
Cognome

Rossi

Turno prenotazione ☒ Pranzo ☐ Cena

CONFERMA ANNULLA

Quando il cliente vuole modificare o cancellare la sua prenotazione deve inserire il codice, il cognome e il turno.



Schermata di conferma cancellazione.