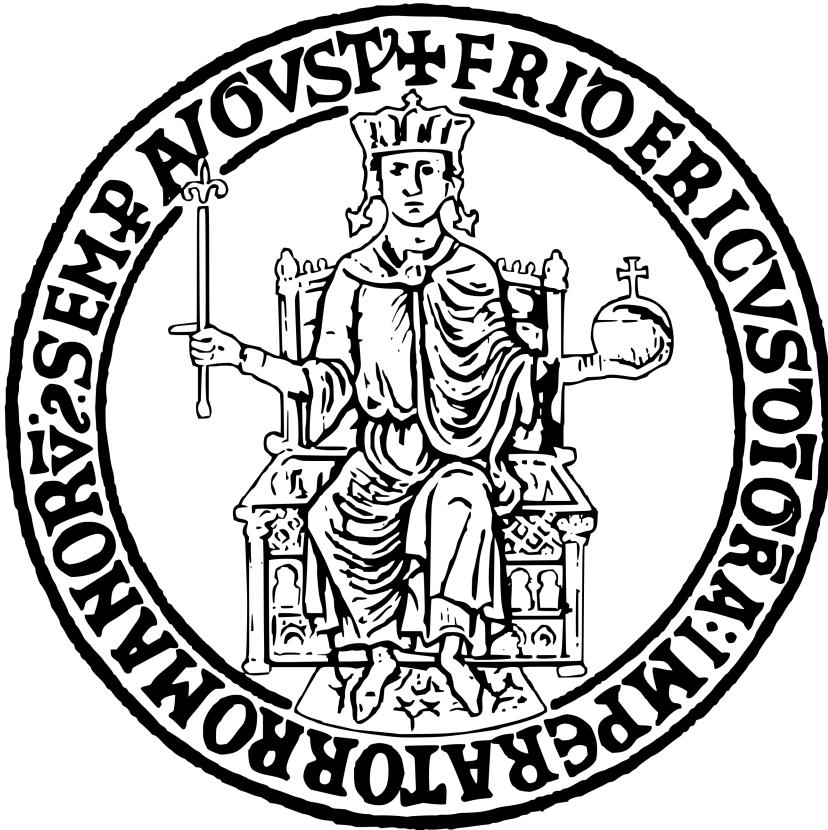


UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II



DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA

INSEGNAMENTO DI INGEGNERIA DEL SOFTWARE

ANNO ACCADEMICO 2022/2023

RATATOUILLE23

AUTORI:

CIRO DE CRISTOFARO
MATRICOLA: N86002370

DOCENTI:

PROF. SERGIO DI MARTINO
PROF. LUIGI LIBERO LUCIO STARACE

Indice

1 Panoramica del Programma	4
1.1 Che cos'è Ratatouille23	4
1.2 Cosa offre?	4
1.3 Cosa contiene questo documento?	4
1.4 Tecnologia utilizzata	4
2 Analisi dei Requisiti	5
2.1 Requisiti dell'applicazione	5
2.1.1 Requisiti funzionali	5
2.1.2 Non funzionali	5
2.1.3 Dominio	5
2.2 Modellazione dei casi d'uso	6
2.2.1 Use Case accesso alla piattaforma	6
2.2.2 Use Case Creazione degli avvisi	6
2.2.3 Use Case visualizzazione degli avvisi	7
2.2.4 Use Case Gestione del Menù	7
2.2.5 Use Case Creazione Categoria	8
2.2.6 Use Case Gestione Categoria	8
2.2.7 Use Case Gestione della sala	9
2.2.8 Use Case Gestione dell'ordinazione creato	9
2.2.9 Use Case Creazione Utenza	10
2.3 Tabelle di CockBurn dei casi d'uso	11
2.4 Mock-up	15
2.4.1 MK_1 Login	16
2.4.2 MK_3 HomePage Addetto Alla Sala	17
2.4.3 MK_4 Registrazione Ordini	18
2.4.4 MK_5 Creazione Avviso	19
2.4.5 MK_6 Visualizza Avviso	20
2.4.6 MK_7 Visualizza Elenco Avvisi	21
2.4.7 MK_9 Creazione Utenza	22
2.4.8 MK_11 Menu Laterale Addetto Alla Sala	23
2.4.9 MK_11 Menu Laterale Addetto alla Cucina	24
2.4.10 MK_11 Menu Laterale Amministratore	25
2.4.11 MK_11 Menu Laterale Supervisore	26
2.4.12 MK_13 HomePage Amministratore	27
2.4.13 MK_14 HomePage Supervisore	28
2.5 Glossario	29
3 Specifica dei Requisiti	30
3.1 Classi, oggetti e relazioni di analisi	30
3.1.1 Classi ed entità	30
3.1.2 Class diagram di Analisi	31
3.1.3 Class diagram di Analisi - Autenticazione	31
3.1.4 Class diagram di Analisi - Gestione menu - Aggiungi Elemento	31
3.1.5 Class diagram di Analisi - Visualizzazione Avviso	32
3.1.6 Class diagram di Analisi - Gestione sala - Registrazione Ordine	32
3.1.7 Class diagram di Analisi - Definizione Ordine Categorie	32
3.1.8 Class diagram di Analisi - Gestione di una Categoria	33
3.1.9 Class diagram di Analisi - Creazione Utenza	33
3.1.10 Class diagram di Analisi - Creazione Categoria	33
3.1.11 Class diagram di Analisi - Creazione Avviso	34
3.1.12 Sequence Diagram di analisi	35
3.1.13 Sequence Diagram di analisi - Creazione Categoria	35
3.1.14 Sequence Diagram di analisi - Creazione Utenza	35
3.2 Statechart di analisi	36
3.2.1 StateChart di analisi - Autenticazione	36
3.2.2 StateChart di analisi - Gestione Menu	36
3.2.3 StateChart di analisi - Visualizza Avviso	36
3.2.4 StateChart di analisi - Creazione Avviso	37
3.2.5 StateChart di analisi - Creazione Utenza	37
3.2.6 StateChart di analisi - Gestione dell'ordine creato	37
3.2.7 StateChart di analisi - Registrazione ordine	37

4 Design di Sistema	38
4.1 Analisi Architetturale	38
4.1.1 Descrizione architettura Cloud	39
4.1.2 Descrizione del server	40
4.1.3 Descrizione API REST	40
4.1.4 Descrizione del Client	42
4.1.5 Classi di supporto	43
4.2 Diagramma delle classi di design	44
4.2.1 Class diagram di design - Login	44
4.2.2 Class diagram di design - Primo Accesso	44
4.2.3 Class diagram di design - Creazione Utenza	45
4.2.4 Class diagram di design - Creazione Avviso	45
4.2.5 Class diagram di design - Gestione Menù	46
4.2.6 Class diagram di design - Gestione Categoria	46
4.2.7 Class diagram di design - Aggiungi Elemento alla Categoria	47
4.2.8 Class diagram di design - Registrazione ordine	47
4.3 Class diagram di design - Gestione dell'ordine creato	48
4.4 Diagramma di stato di design	49
4.4.1 Creazione Avviso	49
4.5 Diagramma di sequenza di design	50
4.5.1 Diagramma di sequenza di design - Creazione Utenza	50
4.5.2 Diagramma di sequenza di design - Creazione Avviso	51
4.5.3 Diagramma di sequenza di design - Visualizza Avviso	52
4.5.4 Diagramma di sequenza di design - Registrazione Ordine	53
5 Codice xUnit	54
5.1 Testing BlackBox	54
5.1.1 Testing BlackBox aggiungiElemento	54
5.1.2 Testing BlackBox creaAvviso	62
5.2 Testing WhiteBox	67
5.2.1 Testing WhiteBox creaCategoriaConElemento	67
5.2.2 Testing WhiteBox reImpostaPassword	76

Page blank

1 Panoramica del Programma

La società SoftEngUniNA ha commissionato al gruppo progetto INGSW2223_V_06 la realizzazione di un sistema informatico multi-piattaforma a commercializzare un applicativo software chiamato **Ratatouille23**

1.1 Che cos'è Ratatouille23

Ratatouille23 è un servizio finalizzato al supporto, alla gestione e all'operatività di attività di ristorazione.

1.2 Cosa offre?

L'applicativo nella sua prima versione di lancio offrirà le seguenti funzionalità richieste dal committente del progetto:

- Login degli utenti mediante email e password sulla piattaforma.
- Possibilità di modificare il menù, con l'aggiunta o l'eliminazione di elementi e la possibilità di ordinare le diverse categorie o elementi.
- Possibilità di creare degli avvisi
- Possibilità di registrare delle ordinazione,
- Possibilità di gestire l'ordinazione creato, con la possibilità di aggiungere, rimuovere, aumentare o diminuire la quantità degli elementi richiesti
- Possibilità di visualizzare e/o nascondere gli avvisi ricevuti
- Possibilità di creare utenze per il personale

1.3 Cosa contiene questo documento?

La documentazione contiene:

- Documento dei Requisiti Software
- Documento di Design del sistema
- Testing

1.4 Tecnologia utilizzata

L'applicativo è stato sviluppato interamente in Android utilizzando il Linguaggio di programmazione **Object-Oriented**¹, è stato affiancato ad un Backend distribuito come container Docker e che sfrutta tecnologie all'avanguardia come servizi di public Cloud Computing come **Azure** o **AWS**, al fine di massimizzare la scalabilità e flessibilità del sistema.

¹sviluppato interamente in Java

2 Analisi dei Requisiti

2.1 Requisiti dell'applicazione

2.1.1 Requisiti funzionali

Vengono ora elencati i requisiti funzionali ovvero tutti i servizi che il sistema deve offrire:

- **Accesso alla piattaforma:** Autenticazione tramite le credenziali di un account Ratatouille23 mediante utilizzo di un email e password valide e al primo accesso di re-impostare la password.
- **Creazione utenze:** Il sistema deve offrire la possibilità all'amministratore di poter creare utenze per i propri dipendenti.
- **Gestione della sala:** Il sistema deve offrire la possibilità all'addetto della sala deve poter registrare le ordinazioni e/o gestire le ordinazioni create in precedenza con la possibilità di aggiungere, rimuovere o modificare la quantità degli elementi desiderati dai clienti.
- **Creazione degli avvisi:** Il sistema deve offrire la possibilità all'amministratore o supervisore di creare avvisi.
- **Visualizzazione degli avvisi:** Il sistema deve offrire la possibilità di visualizzare gli avvisi e marcarli con "visualizzato" e/o nasconderlo.
- **Gestione del Menù:** Il sistema deve offrire la possibilità all'amministratore o al supervisore di poter personalizzare il menù creando e/o eliminare elementi del menu e poter organizzare gli elementi in categorie e definire l'ordine con cui gli elementi compaiono nel menù.

2.1.2 Non funzionali

Vengono ora elencati i requisiti non funzionali ovvero tutti i vincoli sui servizi offerti dal sistema:

- **Scalabilità:** Il sistema deve avere un back-end in cloud scalabile e flessibile per adattarsi a frequenze di accesso elevate.
- **Policy password:** Il sistema deve forzare l'utente ad inserire una password di almeno 8 caratteri contenente numeri, almeno una lettera maiuscola ed una minuscola e un simbolo speciale.
- **Adattabilità al back-end:** Il sistema deve essere adattabile ai possibili cambiamenti del back-end.

2.1.3 Dominio

Vengono ora elencati i requisiti di dominio ovvero i vincoli generali a cui l'applicativo deve attenersi:

- **ISO/IEC:** Il sistema deve essere conforme allo standard ISO/IEC , incentrato sulla protezione dei dati personali nel cloud.
- **GDPR;** Il sistema deve essere conforme al GDPR, incentrato sul trattamento dei dati personali e sulla privacy dell'utente.

2.2 Modellazione dei casi d'uso

2.2.1 Use Case accesso alla piattaforma

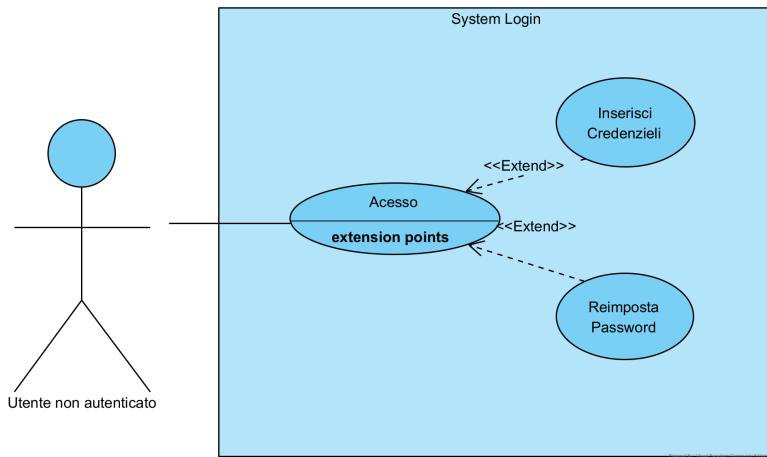


Figura 1: Autenticazione

In questo Use Case viene descritto la fase di autenticazione(Login) sotto il punto di vista di un utente non ancora autenticato.

2.2.2 Use Case Creazione degli avvisi

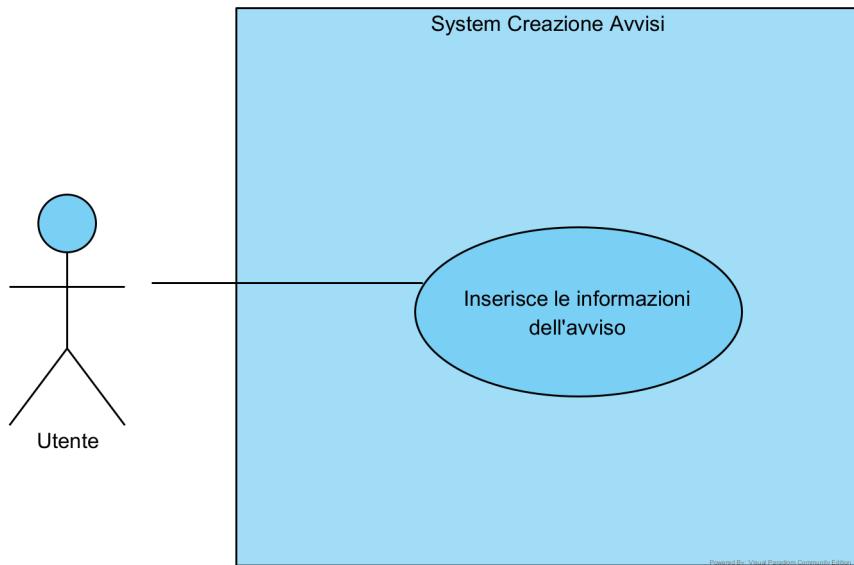


Figura 2: Creazione Avvisi

In questo Use Case viene descritto la fase in cui l'Utente (Amministratore e/o Supervisore) possono creare gli Avvisi.

2.2.3 Use Case visualizzazione degli avvisi

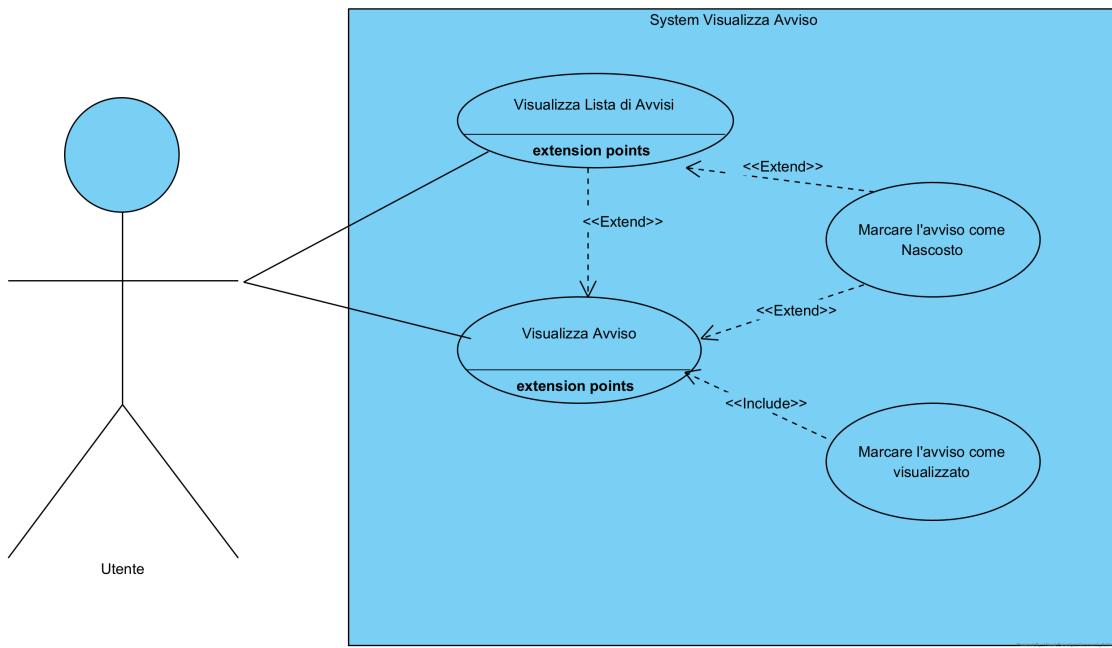


Figura 3: visualizzazione degli avvisi

In questo Use Case viene descritto la fase in cui l'Utente(Amministratore, Supervisore, Addetto alla sala e alla cucina) può visualizzare la lista degli avvisi ricevuti, visualizzare un determinato avviso e/o di nascondere un determinato avviso.

2.2.4 Use Case Gestione del Menù

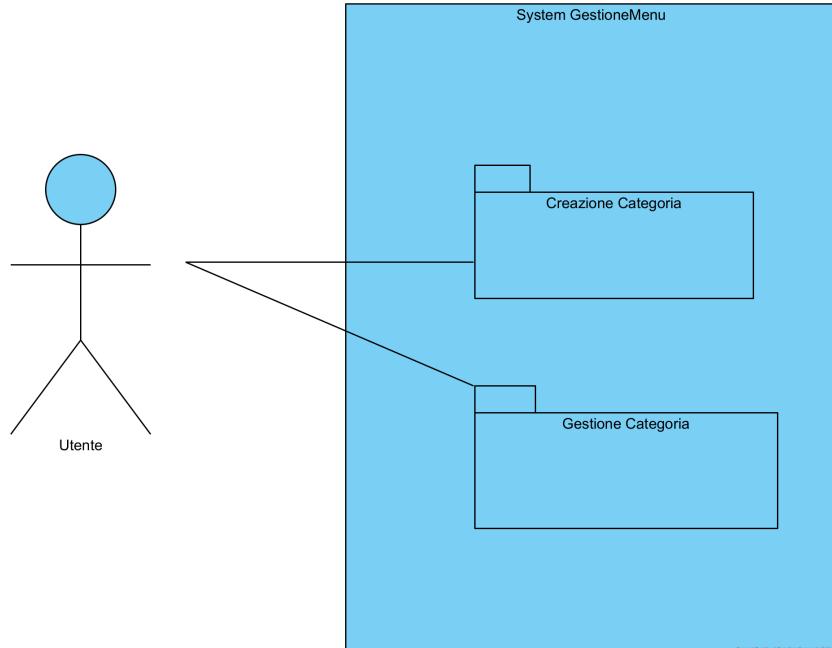


Figura 4: Gestione del Menù

In questo Use Case viene descritto la fase in cui l'Utente (Amministratore o Supervisore) può gestire le categorie con i suoi relativi elementi o di creare una nuova categoria.

2.2.5 Use Case Creazione Categoria

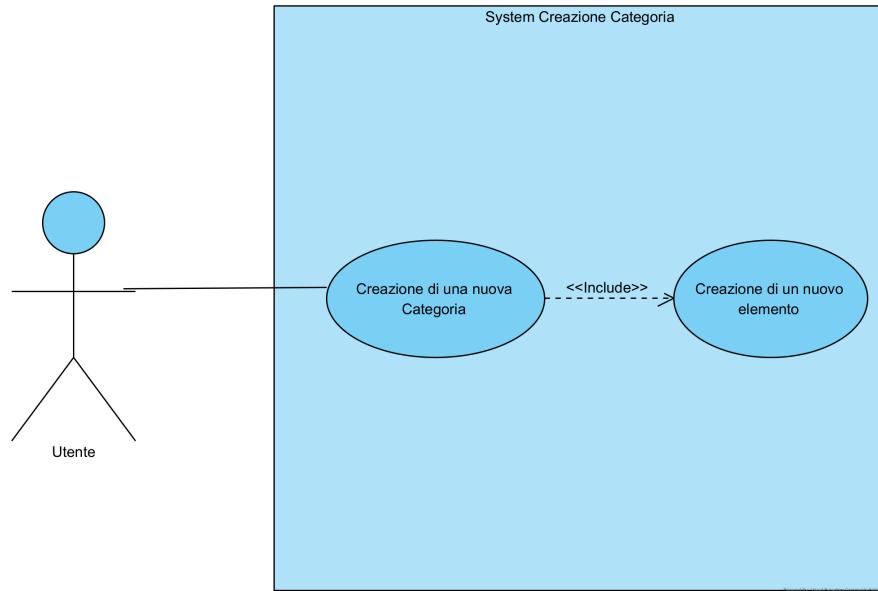


Figura 5: Creazione Categoria

In questo Use Case viene descritto la fase in cui l'Utente(Amministratore o Supervisore) può creare una nuova categoria con un elemento.

2.2.6 Use Case Gestione Categoria

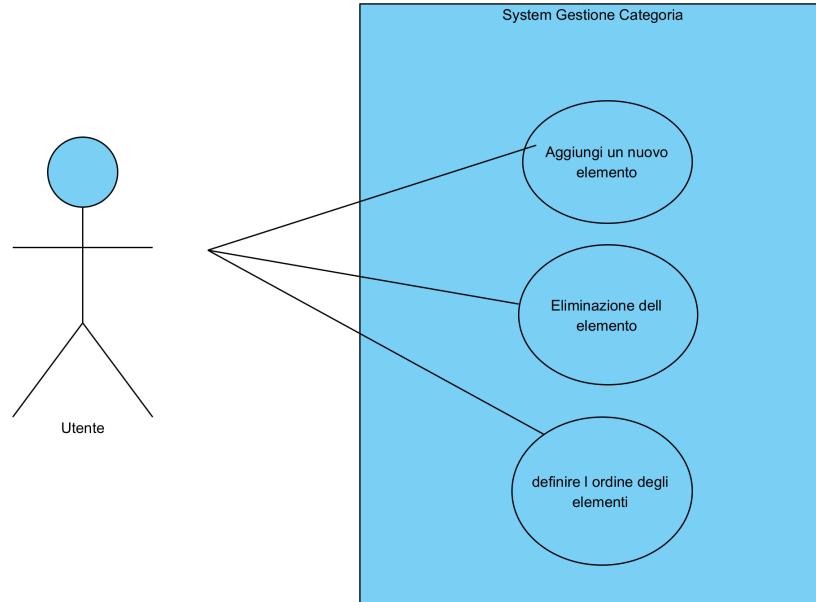


Figura 6: Gestione elementi

In questo Use Case viene descritto la fase in cui l'Utente (Amministratore o Supervisore) può gestire gli elementi di una categoria con la possibilità di aggiungere un nuovo l'elemento, eliminare un elemento dalla categoria e di definire l'ordine un cui appaiono gli elementi

2.2.7 Use Case Gestione della sala

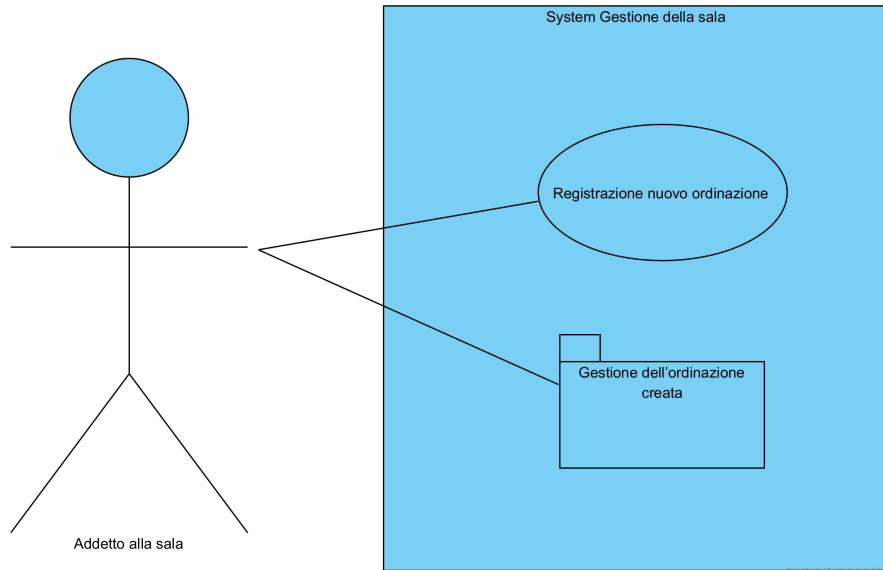


Figura 7: Gestione dell'ordinazione creata

In questo Use Case viene descritto la fase in cui l'Addetto alla sala può creare una nuova ordinazione di un tavolo, inserendo il numero del tavolo e gli elementi desiderati dai clienti o di gestire un ordinazione creata in precedenza.

2.2.8 Use Case Gestione dell'ordinazione creato

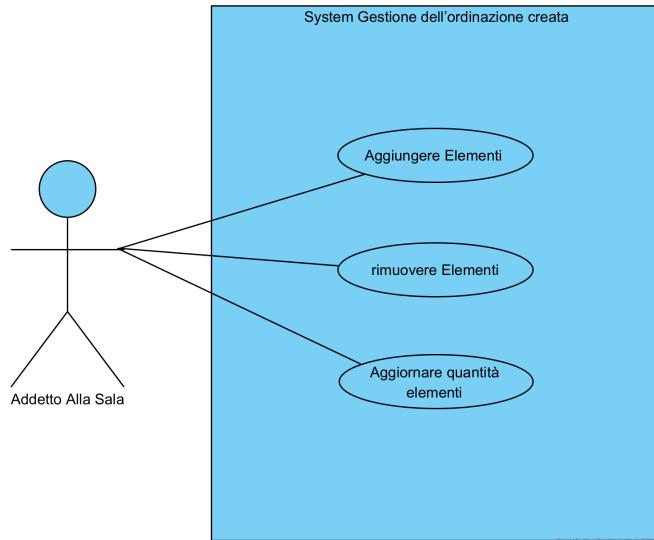


Figura 8: Gestione dell'ordinazione creato

In questo Use Case viene descritto la fase in cui l'addetto alla sala può aggiungere nuovi elementi, rimuovere elementi o aggiornare la quantità degli elementi ordinati in precedenza

2.2.9 Use Case Creazione Utenza

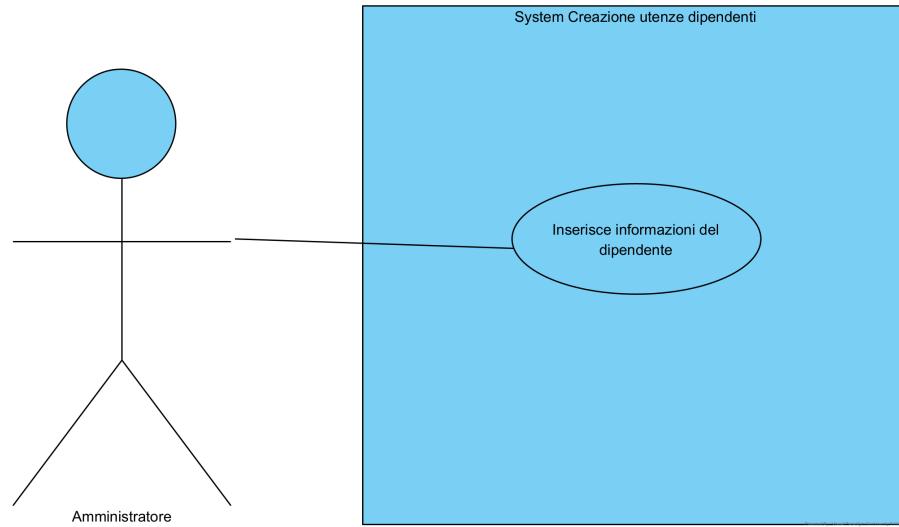


Figura 9: Creazione Utenza

In questo Use Case viene descritto la fase in cui l'Amministratore crea l'utenza per i suoi dipendenti, inserendo i dati personali del dipendente e selezionando il ruolo ad esso assengato.

2.3 Tabelle di CockBurn dei casi d'uso

Use Case #1	Creazione Avviso		
Goal in context	L'amministratore o supervisore vuole creare un avviso		
Precondition	L'utente deve essere autenticato come amministratore o supervisore		
Success End Condition	L'utente riesce a creare un avviso		
Failed End Condition	L'utente non riesce a creare un avviso		
Primary Actor	Utente Autenticato		
Trigger	L'utente preme il pulsante per creare un nuovo avviso		
EXTENSION #1: Il sistema non riesce a collegarsi al server cloud.	Step	Utente Autenticato	Sistema
	1	Clicca sul Pulsante "Crea Avviso" di Mk_14/MK_13	
	2		Mostra MK_5
	3	Inserisce i campi richiesti	
	4	Preme il pulsante "INVIA"	
	5		Mostra pop-up di successo "Avviso Creato con Successo"
EXTENSION #2: Creazione di un avviso esistente nella stessa giornata.	Step	Utente Autenticato	Sistema
	5.a		Mostra pop-up di errore "Server al momento non disponibile"
EXTENSION #3: Il token è scaduto.	Step	Utente Autenticato	Sistema
	5.b		Mostra popup di errore "Avviso già esistente per la data odierna".
	5.d		Torna allo step 3 dello scenario principale
SUBVARIANT #1: Clicca "crea Avviso" in MK_11.	Step	Utente Autenticato	Sistema
			Mostra popup di errore "Sessione utente scaduta. Effettuare il login".
	5.c		Reindirizza alla MK_1

Tabella 1: Creazione Avviso

Use Case #2	Visualizza Avviso		
Goal in context	L'utente vuole visualizzare un avviso		
Precondition	L'utente deve essere autenticato come Amministratore o Supervisore o Addetto alla sala o Addetto alla cucina		
Success End Condition	L'utente riesce a visualizzare un avviso		
Failed End Condition	L'utente non riesce a visualizzare un avviso		
Primary Actor	Utente Autenticato		
Trigger	L'utente preme il pulsante "Notifiche" per visualizzare gli avvisi		
	Step	Utente Autenticato	Sistema
	1	Clicca sul Pulsante "Notifiche" di MK_11	
	2		Mostra MK_7
	3	Clicca su visualizza di un Avviso	
EXTENSION #1: Il sistema non riesce a collegarsi al server cloud.	4		Mostra MK_6
	Step	Utente Autenticato	Sistema
	6.a		Mostra pop-up di errore "Server al momento non disponibile"
EXTENSION #2: Nascondere un avviso.	Step	Utente Autenticato	Sistema
	3.a	Clicca su "Nascondi"	
	4.a		Il sistema nasconde l'avviso
EXTENSION #4: Nessun messaggio da visualizzare	4.a		Torna allo step 2 dello scenario principale
	Step	Utente Autenticato	Sistema
	3.a		Mostra a schermo "Nessun Avviso"
SUBVARIANT #1: L'addetto di sala clicca icona "campanella" di MK_3.	Step	Utente Autenticato	Sistema
	3.c		Torna allo step 2 dello scenario principale

Tabella 2: visualizzazione Avviso

Use Case #3	Registrazione Ordine		
Goal in context	L'addetto alla sala vuole registrazione un ordinazione		
Precondition	L'utente deve essere autenticato come addetto alla sala		
Success End Condition	L'utente riesce a registrare un ordinazione		
Failed End Condition	L'utente non riesce a registrare un ordinazione		
Primary Actor	Utente Autenticato		
Trigger	L'utente clicca il pulsante "Aggiungi Tavolo" per aggiungere un nuovo tavolo		
EXTENSION #1: Il sistema non riesce a collegarsi al server cloud.	Step	Utente Autenticato	Sistema
	1	Clicca sul Pulsante "+" di MK_3 per aggiungere un tavolo	
	2		Mostra MK_4
	3	Inserisce i campi richiesti	
	4	Preme sul pulsante "Salva"	
	5		Mostra pop-up di successo "Ordinazione creata con successo"
EXTENSION #2: Tavolo Occupato.	Step	Utente Autenticato	Sistema
	5.a		Mostra pop-up di errore "Server al momento non disponibile"
EXTENSION #3: Il token è scaduto.	Step	Utente Autenticato	Sistema
	5.b		Mostra pop-up di errore "Tavolo già occupato, controllare il numero tavolo"
SUBVARIANT #1: Clicca sull'icona "Aggiung Tavolo" di MK_11.	Step	Utente Autenticato	Sistema
	1.c		Mostra popup di errore "Sessione utente scaduta. Effettuare il login".
	2.c		Reindirizza alla LoginPageMockup
	Step	Utente Autenticato	Sistema
	3.d		Torna allo step 3 dello scenario principale

Tabella 3: Registrazione Ordine

Use Case #4	Creazione Utenza		
Goal in context	L'amministratore vuole creare l'utenza per i dipendenti		
Precondition	L'utente deve essere autenticato come amministratore		
Success End Condition	L'utente riesce a creare l'utenza per il personale		
Failed End Condition	L'utente non riesce a creare l'utenza per il personale		
Primary Actor	Utente Autenticato		
Trigger	L'utente clicca il pulsante "Aggiungi Personale" per creare l'utenza al personale		
	Step	Utente Autenticato	Sistema
	1	Clicca sul Pulsante "Aggiungi Personale" di MK_13	
	2		Mostra MK_9
	3	Inserisce le informazioni del personale	
	4	Preme il pulsante "Crea"	
	5		Mostra pop-up di successo "Utenza creata con successo"
EXTENSION #1: Il sistema non riesce a collegarsi al server cloud.	Step	Utente Autenticato	Sistema
	5.a		Mostra pop-up di errore "Server al momento non disponibile"
EXTENSION #3: Email non corretta.	Step	Utente Autenticato	Sistema
	5.b		Mostra pop-up di errore "Formato email non valida" Torna allo step 5 dello scenario Principale
EXTENSION #4: Email già esistente	Step	Utente Autenticato	Sistema
	5.c		Mostra pop-up di errore "Email già in uso" Torna allo step 3 dello scenario Principale
EXTENSION #5: Formato Password non corretto	Step	Utente Autenticato	Sistema
	5.d		Mostra pop-up di errore "Password non conforme"
	5.d		Torna allo step 3 dello scenario Principale
EXTENSION #6: Il token è scaduto.	Step	Utente Autenticato	Sistema
	5.e		Mostra popup di errore "Sessione utente scaduta. Effettuare il login".
	5.e		Mostra MK_1
SUBVARIANT #1: Clicca sull'icona "Aggiungi personale" di MK_11.	Step	Utente Autenticato	Sistema
	3.g		Torna allo step 3 dello scenario principale

Tabella 4: Creazione Utenza

2.4 Mock-up

Di seguito saranno presentati tutti i mock-up di riferimento creati tramite Figma.
N.B. Il design definitivo potrebbe aver subito delle variazioni

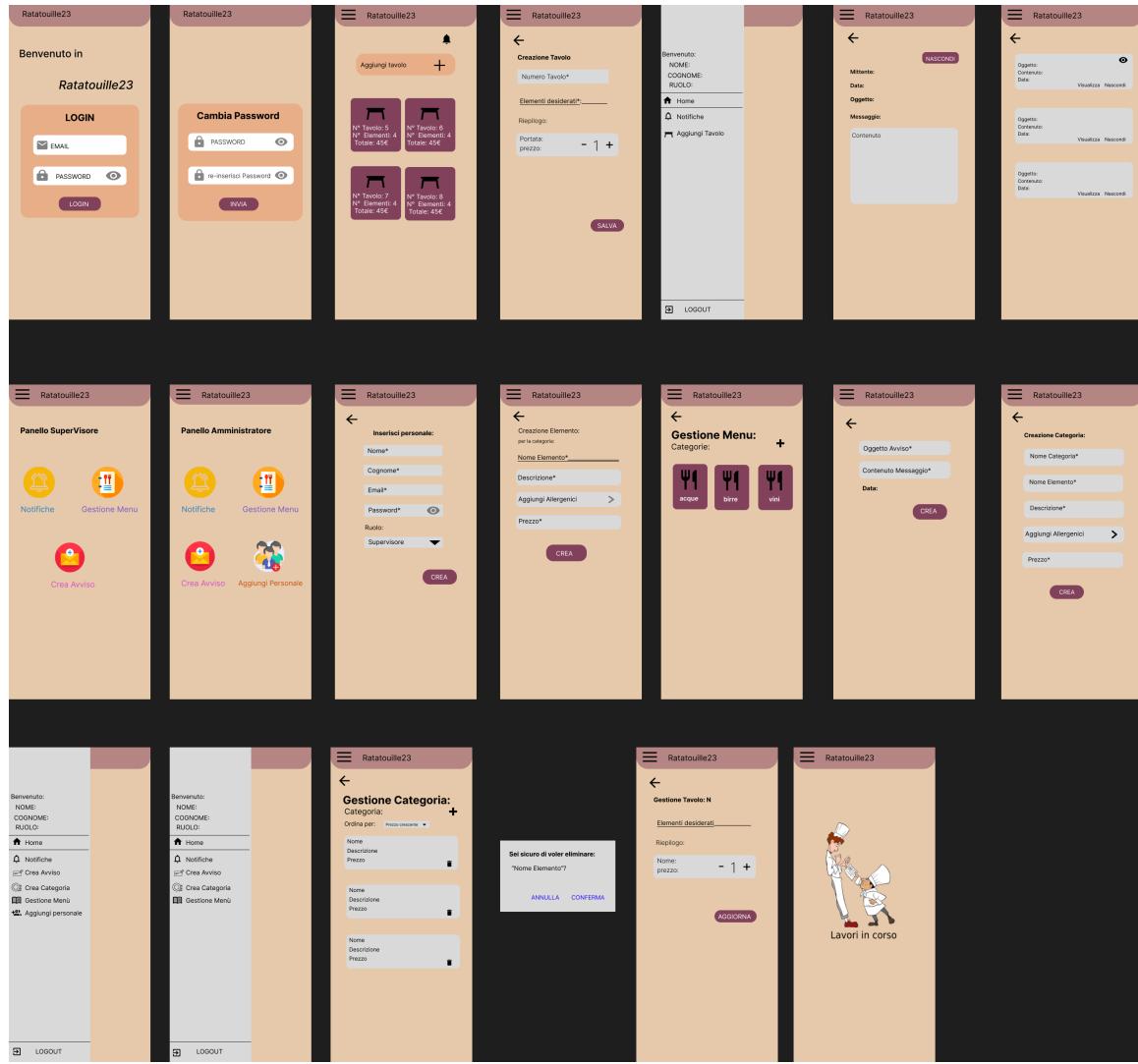


Figura 10: Mock-up - Generale

2.4.1 MK_1 Login

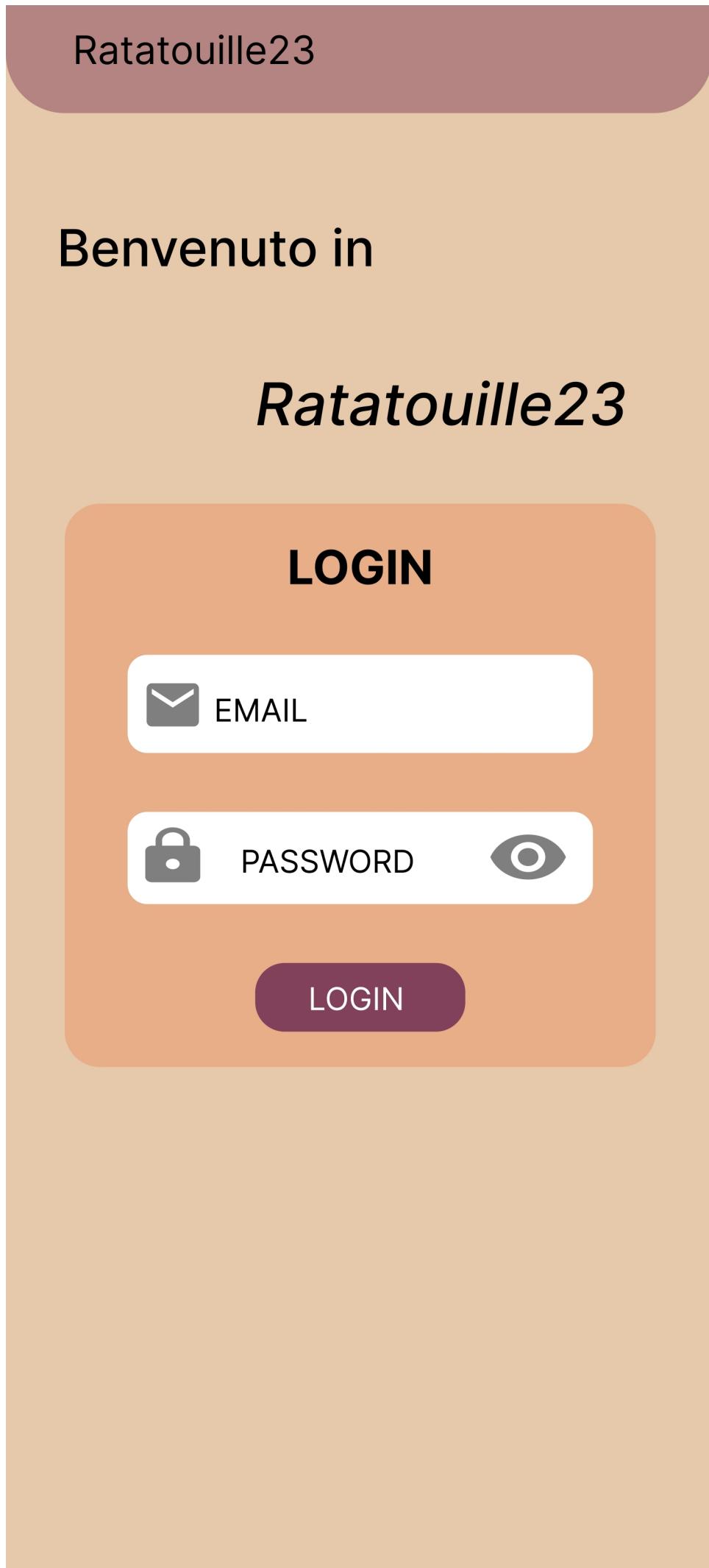


Figura 11: Mock-up - MK_1_login

2.4.2 MK_3 HomePage Addetto Alla Sala



Figura 12: Mock-up - MK_3_HomepageAddettoAllaSala

2.4.3 MK_4 Registrazione Ordini

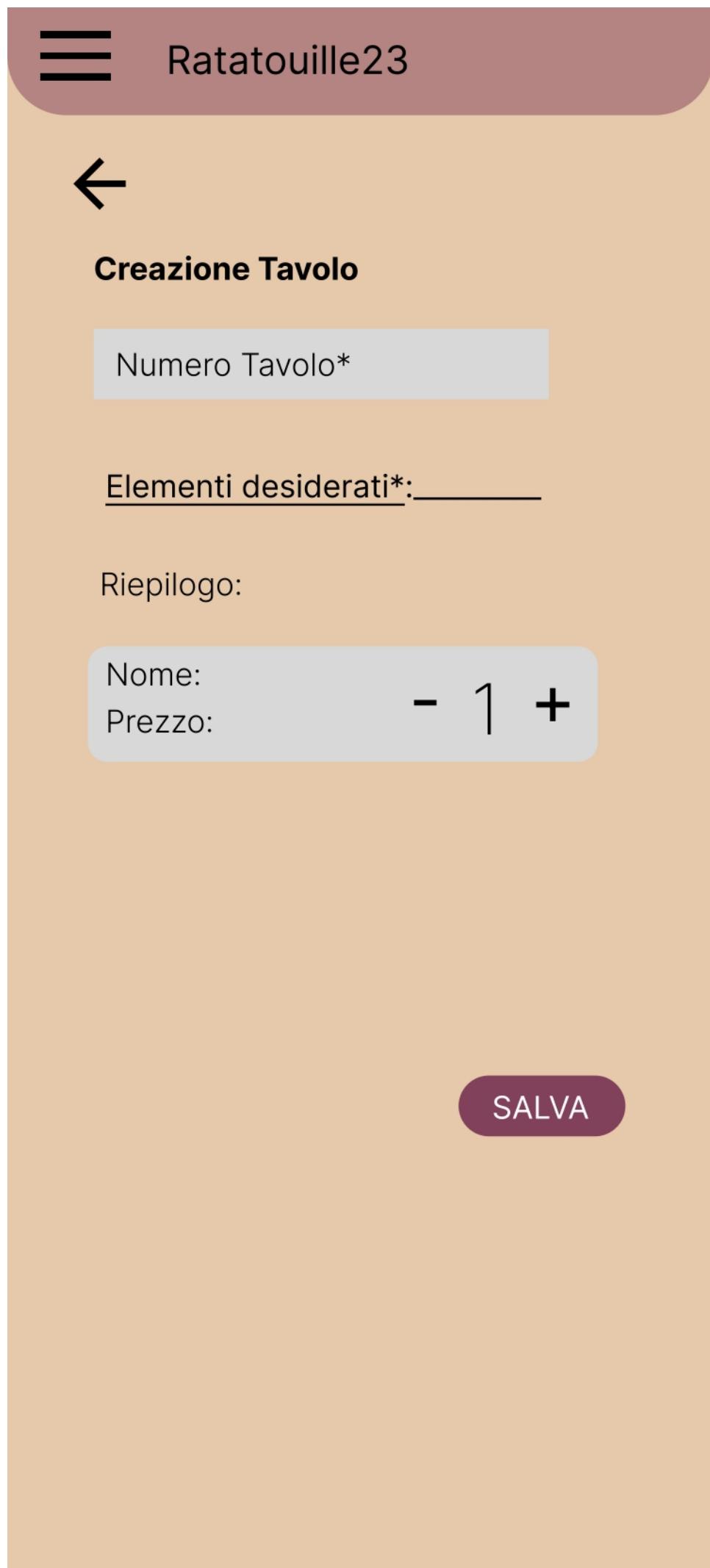


Figura 13: Mock-up - MK_4_Registrazione Ordini

2.4.4 MK_5 Creazione Avvisi

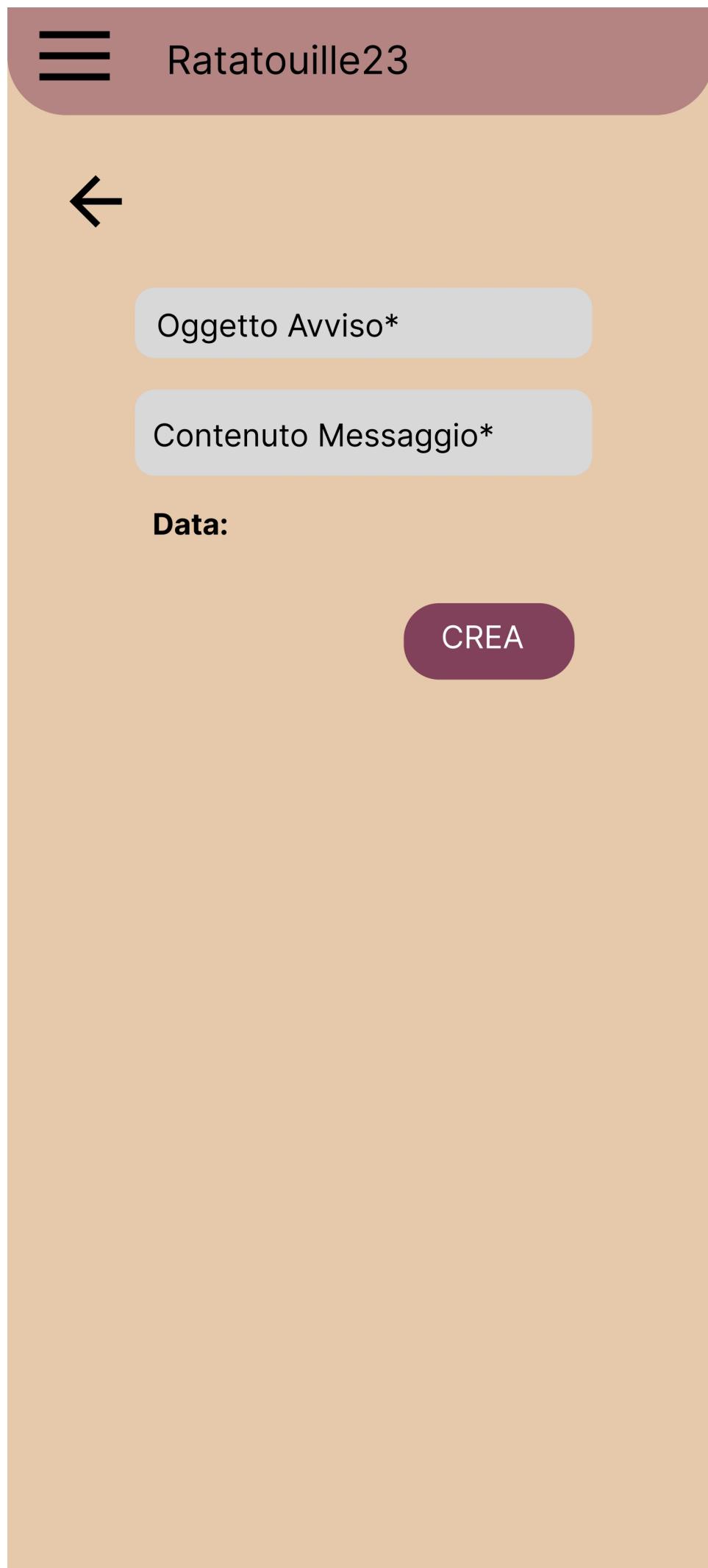


Figura 14: Mock-up - MK_5_CreazioneAvviso

2.4.5 MK_6 Visualizza Avviso

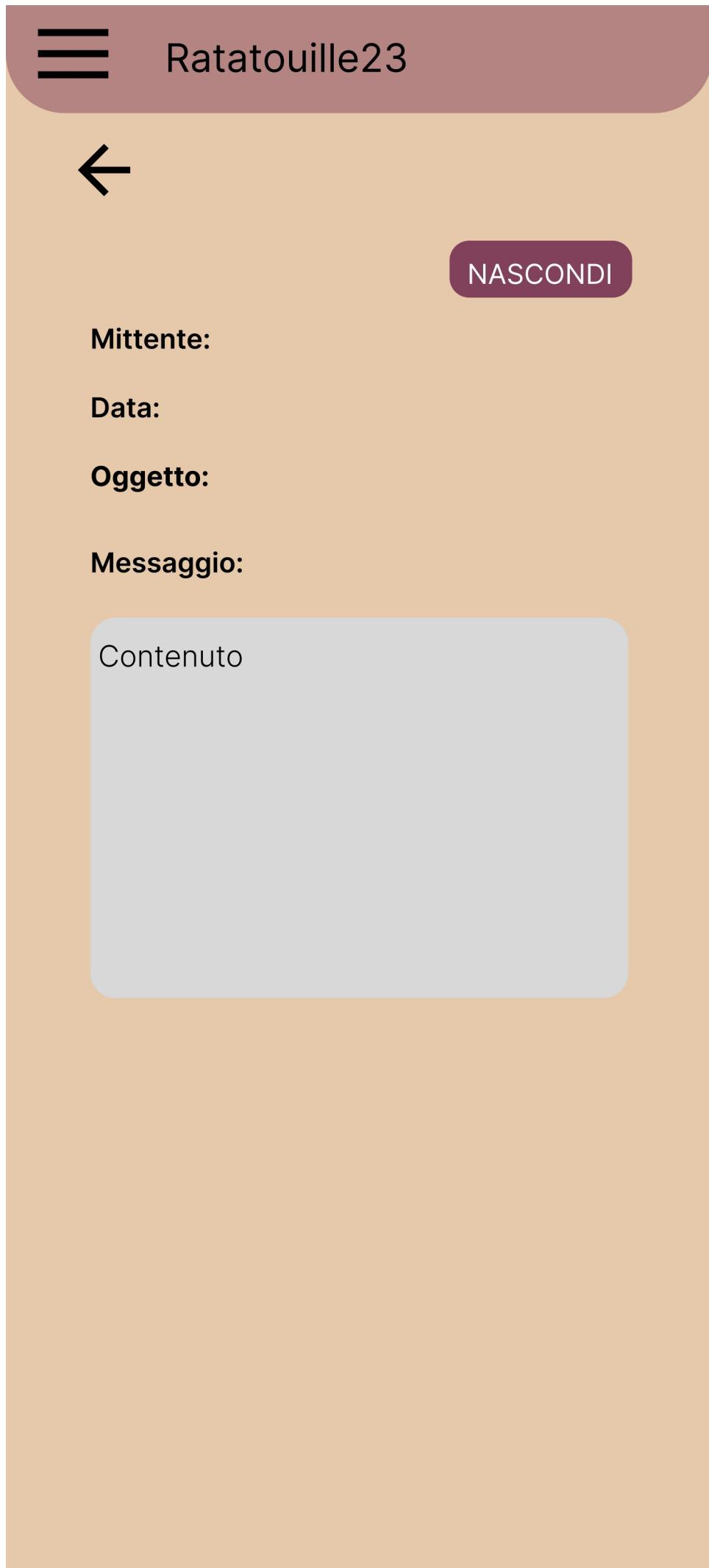


Figura 15: Mock-up - MK_6_Visualizza Avviso

2.4.6 MK_7 Visualizza Elenco Avvisi

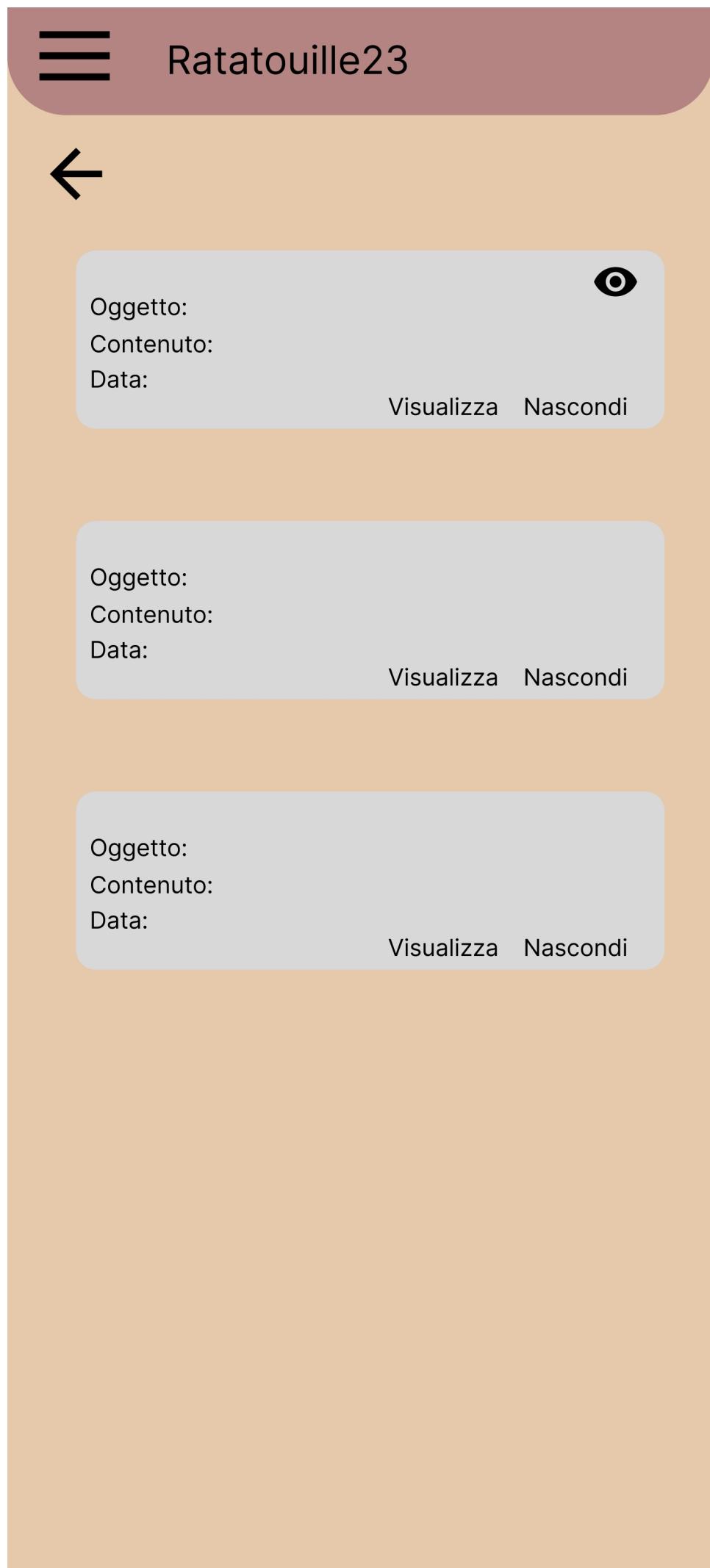


Figura 16: Mock-up - MK_7_Visualizza Avviso

2.4.7 MK_9 Creazione Utenza

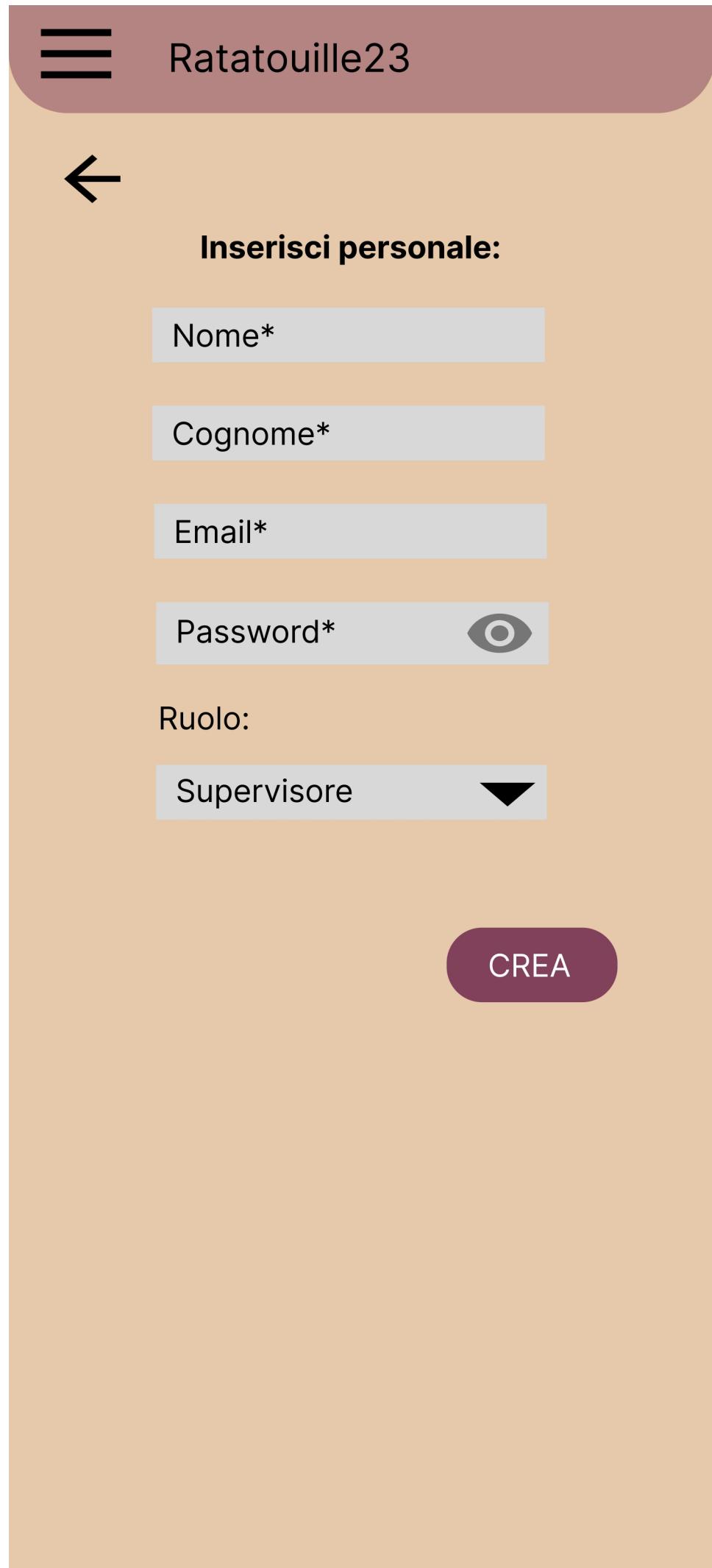


Figura 17: Mock-up - MK_9_Creazione Utenza

2.4.8 MK_11 Menu Laterale Addetto Alla Sala



Figura 18: Mock-up - MK_11_Menu Laterale Addetto Alla Sala

2.4.9 MK_11 Menu Laterale Addetto alla Cucina



Figura 19: Mock-up - MK_11_ Menu Laterale Addetto alla Cucina

2.4.10 MK_11 Menu Laterale Amministratore

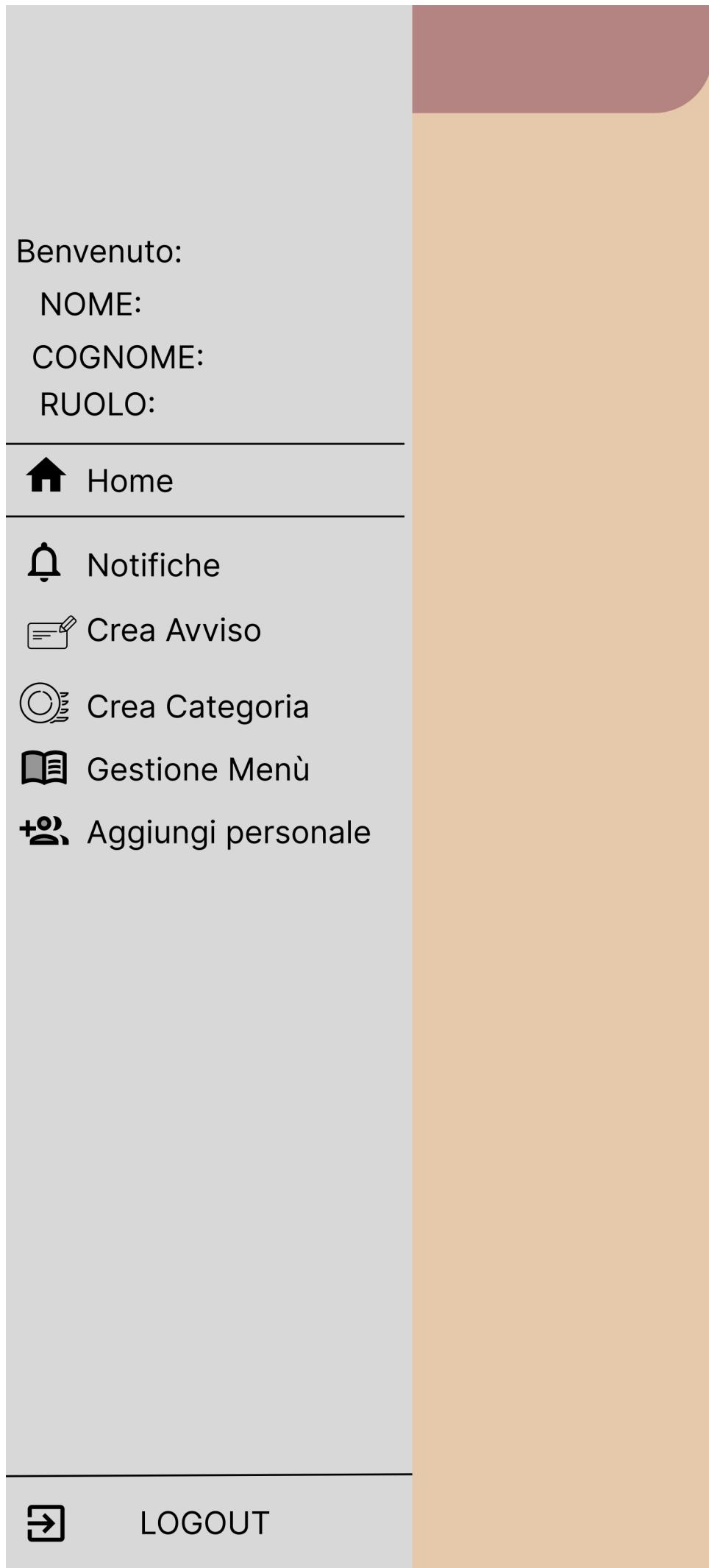


Figura 20: Mock-up - MK_11_ Menu Laterale Amministratore

2.4.11 MK_11 Menu Laterale Supervisore

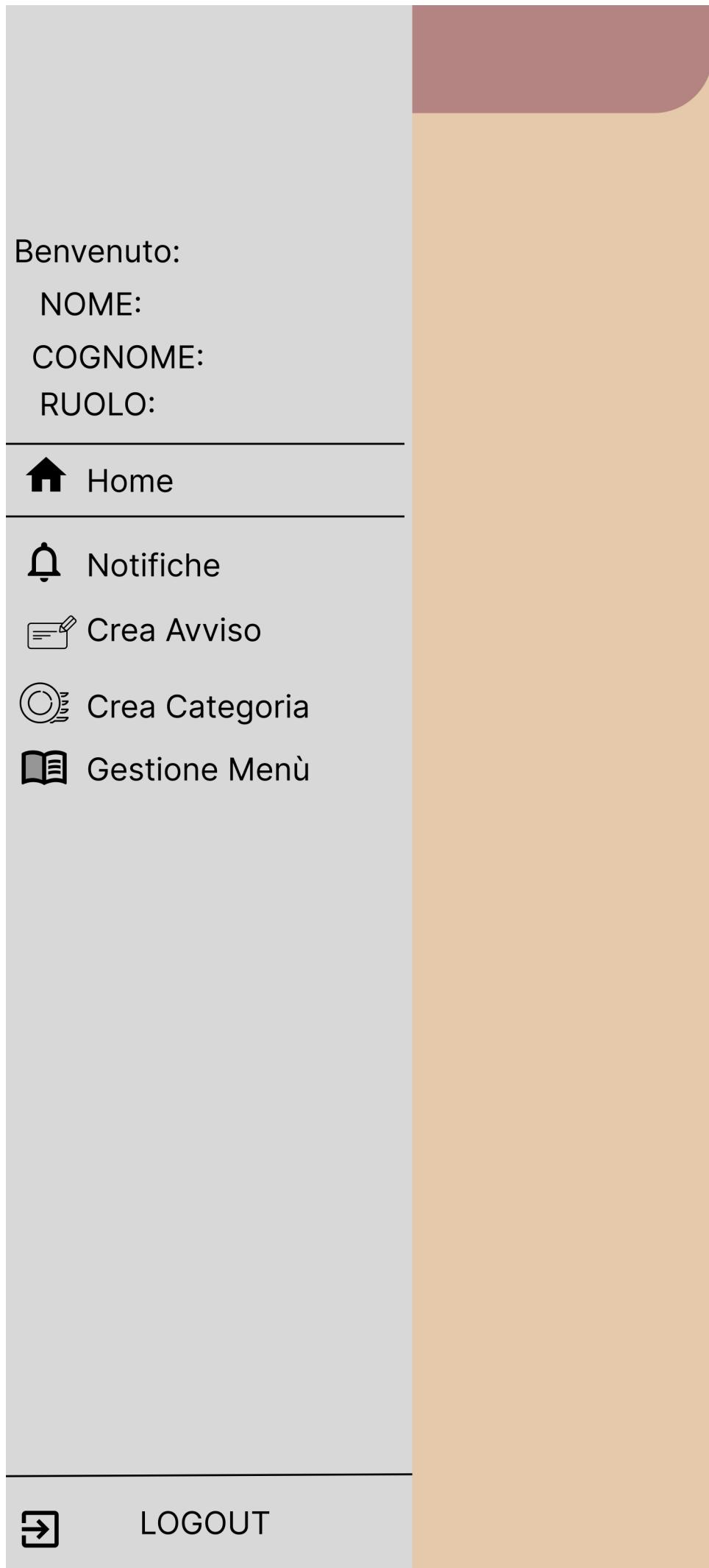


Figura 21: Mock-up - MK_11_ Menu Laterale Supervisore

2.4.12 MK_13 HomePage Amministratore

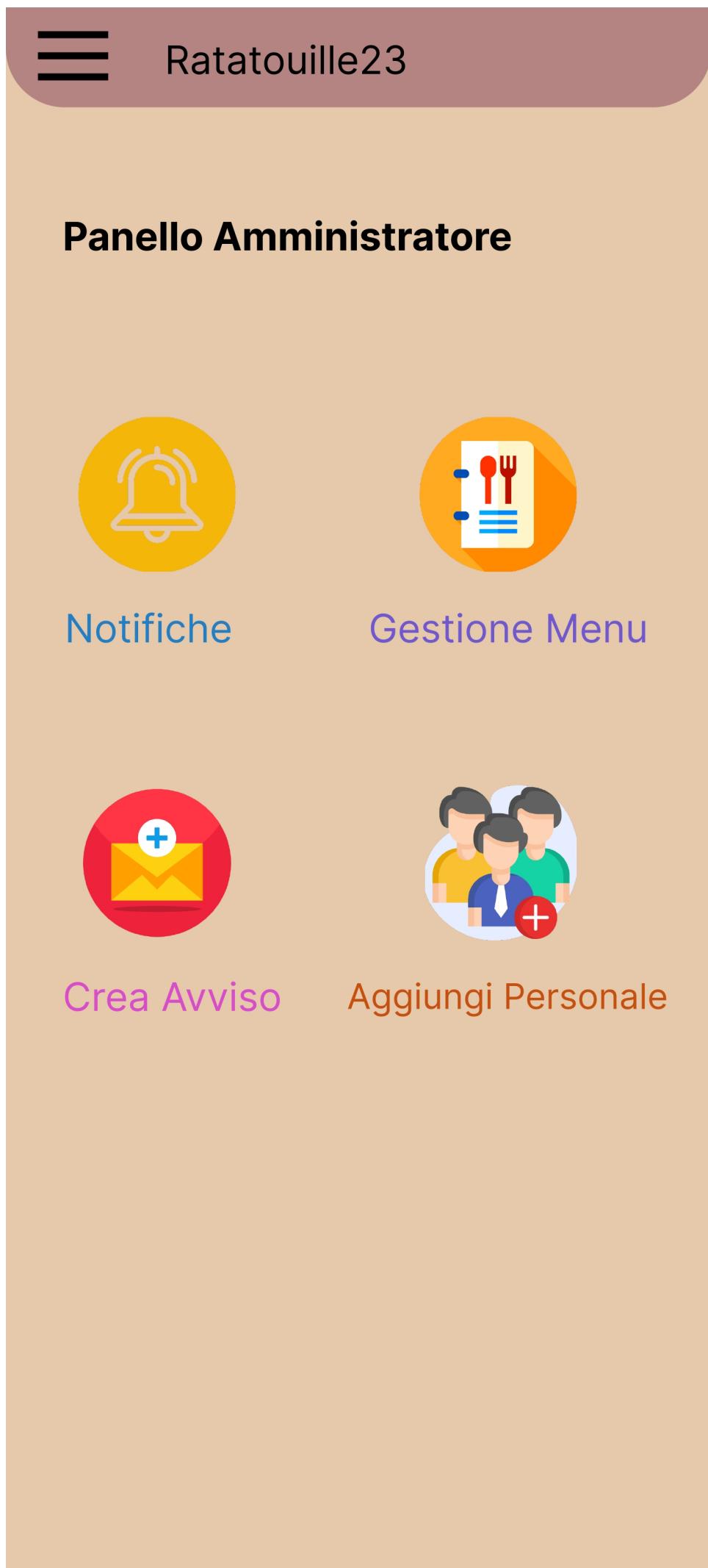


Figura 22: Mock-up - MK_13_ HomePage Amministratore

2.4.13 MK_14 HomePage Supervisore

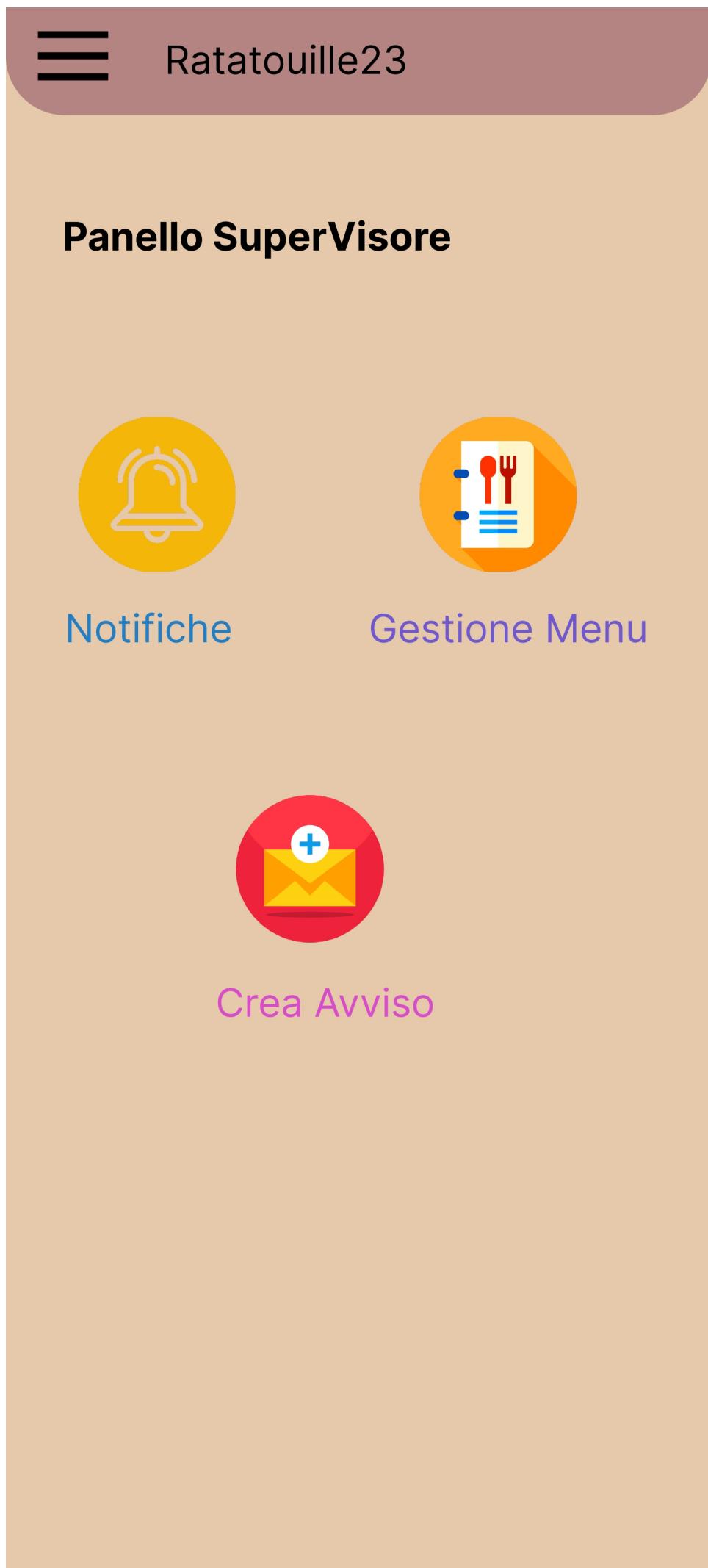


Figura 23: Mock-up - MK_14_ HomePage Supervisore

2.5 Glossario

Termine	Descrizione
Mock-up	Modello grafico di un prodotto che possono essere utilizzati per comunicare le idee ai team di sviluppo e ai clienti.
Use case Diagram	Mostrano le interazioni e descrivono le funzionalità di un sistema dal punto di vista dell'utente .
StateChart Diagram	Descrivono il comportamento dinamico di un sistema o di un oggetto.
Sequence Diagram	Mostrano come gli oggetti interagiscono tra loro per eseguire uno scenario Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate
Class Diagram	Sono uno dei tipi di diagrammi che possono comparire in un modello UML. Essi consentono di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni fra questi.
Tabelle di CockBurn	Tabelle di rappresentazione di un caso d'uso
Android	Sistema operativo open-source
Design Pattern	Una soluzione progettuale generale ad un problema ricorrente.
Server	Sono macchine che offrono dei servizi ai client che li richiedono
Client	Sono macchine che richiedono servizi ai server
Testing	Processo di verifica che un prodotto software funzioni come previsto
Black-Box	È una tipologia di Testing, in cui si concentra sul comportamento del software invece del codice
White-Box	È una tipologia di Testing, che si basa sulla conoscenza della struttura interna del software oltre che alle sue funzionalità
API REST	Sono un insieme di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi.
Framework	È un ambiente di sviluppo software che fornisce un set di funzioni e funzionalità comuni che possono essere utilizzate per sviluppare applicazioni software.
JUnit	Framework di unit testing per la lingua di programmazione Java
Mockito	È un framework di mocking per Java. Il mocking è una tecnica di test per simulare il comportamento di un oggetto.
Docker	È una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità.

3 Specifica dei Requisiti

In questa sezione, verranno analizzati i requisiti del sistema che sono ancora in fase di analisi. In particolare, saranno considerati tre diagrammi fondamentali sia per la fase di analisi che per la fase di design.

I diagrammi presi in considerazione sono:

- Class diagram - diagramma delle classi
- Sequence diagram - analisi di sequenza dell'interazione tra oggetti in un singolo scenario
- Statechart di analisi - diagramma degli stati che descrivere il comportamento di un oggetti o classi in termini di stato

3.1 Classi, oggetti e relazioni di analisi

3.1.1 Classi ed entità

Adesso verranno riportate le entità che sono state individuate durante le fasi di analisi.

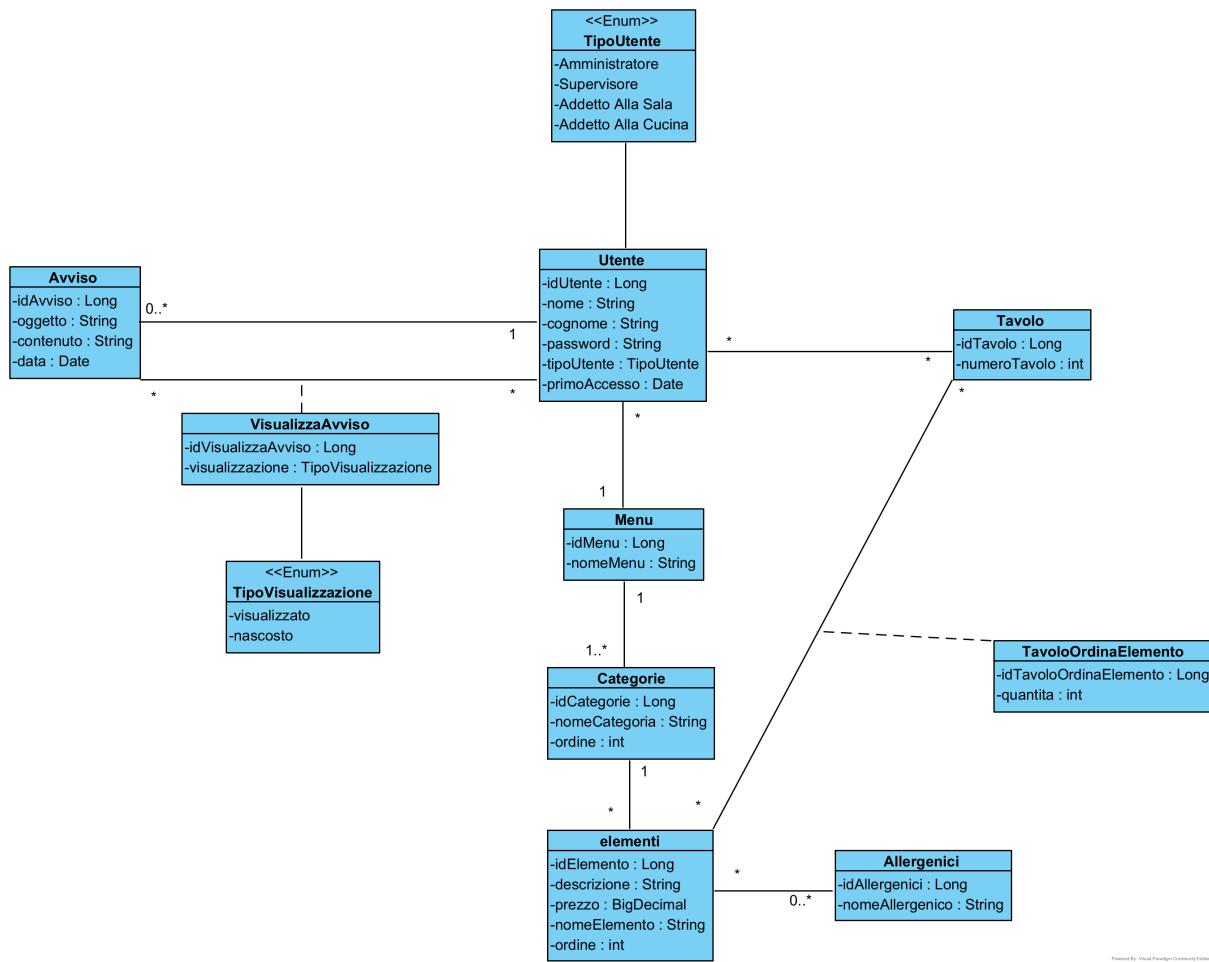


Figura 24: Class Diagram Generale

3.1.2 Class diagram di Analisi

Di seguito vengono riportati i class diagram di analisi suddivisi per funzionalità, al fine di renderli più gestibili e comprensibili.

3.1.3 Class diagram di Analisi - Autenticazione

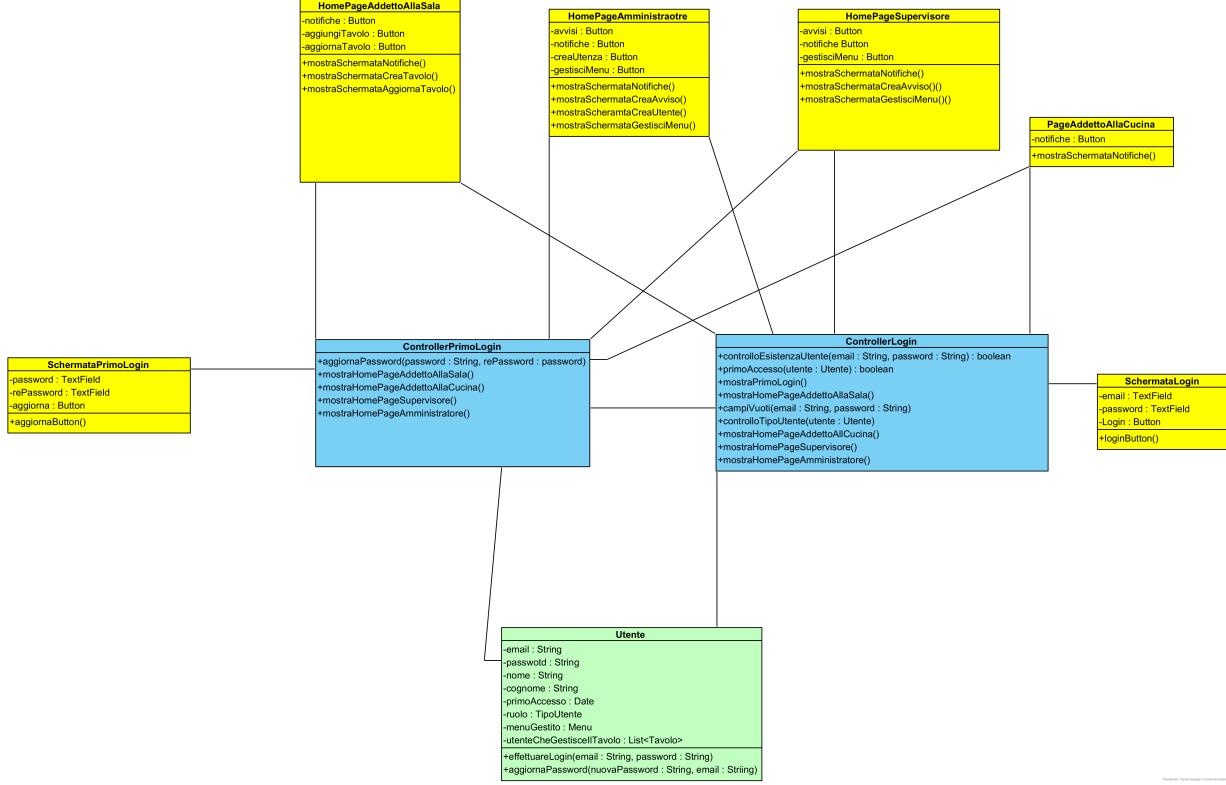


Figura 25: Class Diagram di Analisi - Autenticazione

3.1.4 Class diagram di Analisi - Gestione menu - Aggiungi Elemento

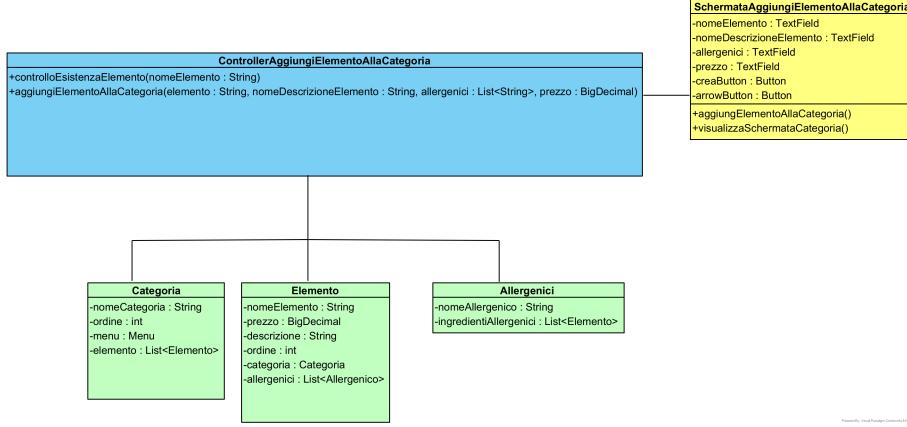


Figura 26: Class Diagram di Analisi - Aggiungi elemento

3.1.5 Class diagram di Analisi - Visualizzazione Avviso

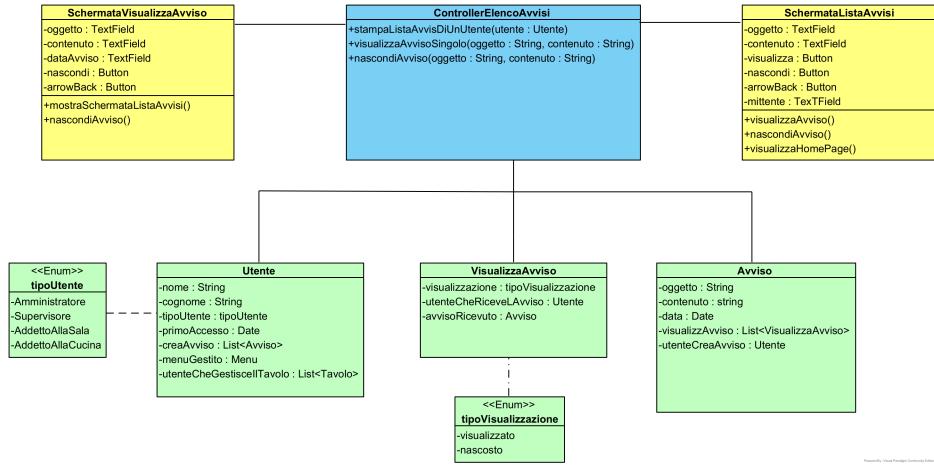


Figura 27: Class Diagram di Analisi Visualizza Avviso

3.1.6 Class diagram di Analisi - Gestione sala - Registrazione Ordine

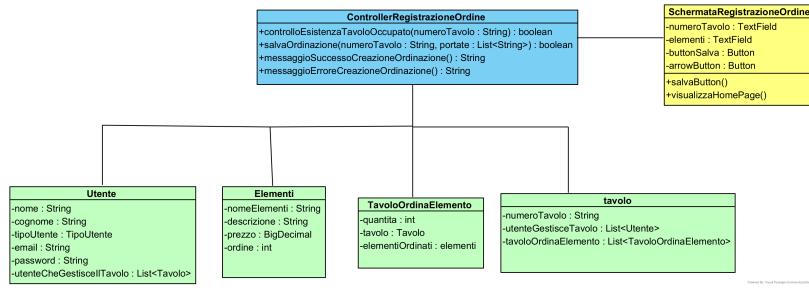


Figura 28: Class Diagram di Analisi Registrazione Ordine

3.1.7 Class diagram di Analisi - Definizione Ordine Categorie

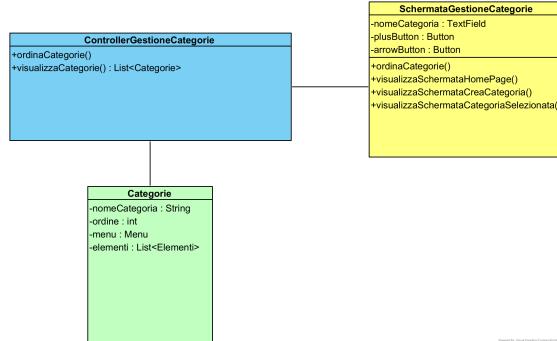


Figura 29: Class Diagram di Analisi - Definizione Ordine Categorie

3.1.8 Class diagram di Analisi - Gestione di una Categoria

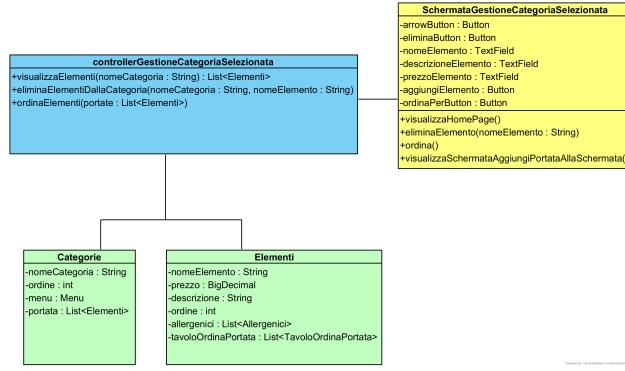


Figura 30: Class Diagram di Analisi - Gestione elementi di una determinata categoria

3.1.9 Class diagram di Analisi - Creazione Utenza

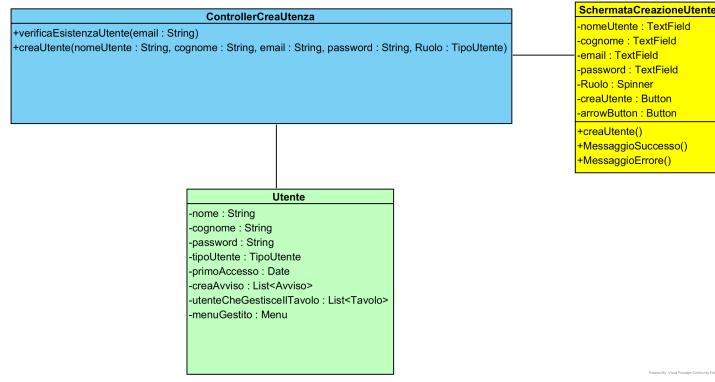


Figura 31: Class Diagram di Analisi Definizione creazione Utenza

3.1.10 Class diagram di Analisi - Creazione Categoria

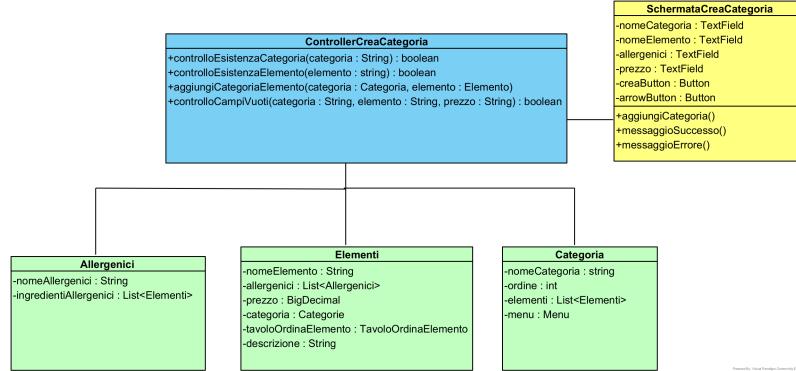


Figura 32: Class Diagram di Analisi - Creazione categoria

3.1.11 Class diagram di Analisi - Creazione Avvisi

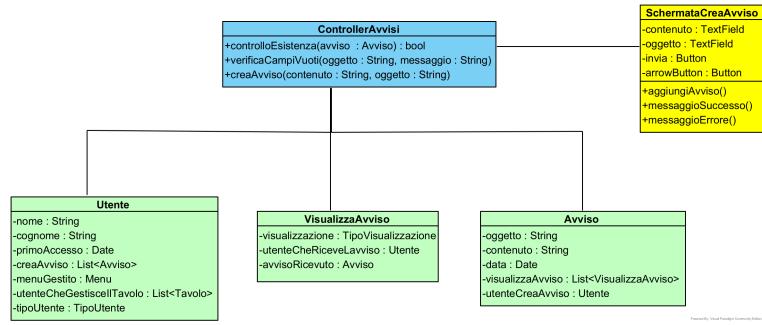


Figura 33: Class Diagram di Analisi - Creazione Avvisi

3.1.12 Sequence Diagram di analisi

Adesso verranno riportato i Sequence diagram di analisi.

3.1.13 Sequence Diagram di analisi - Creazione Categoria

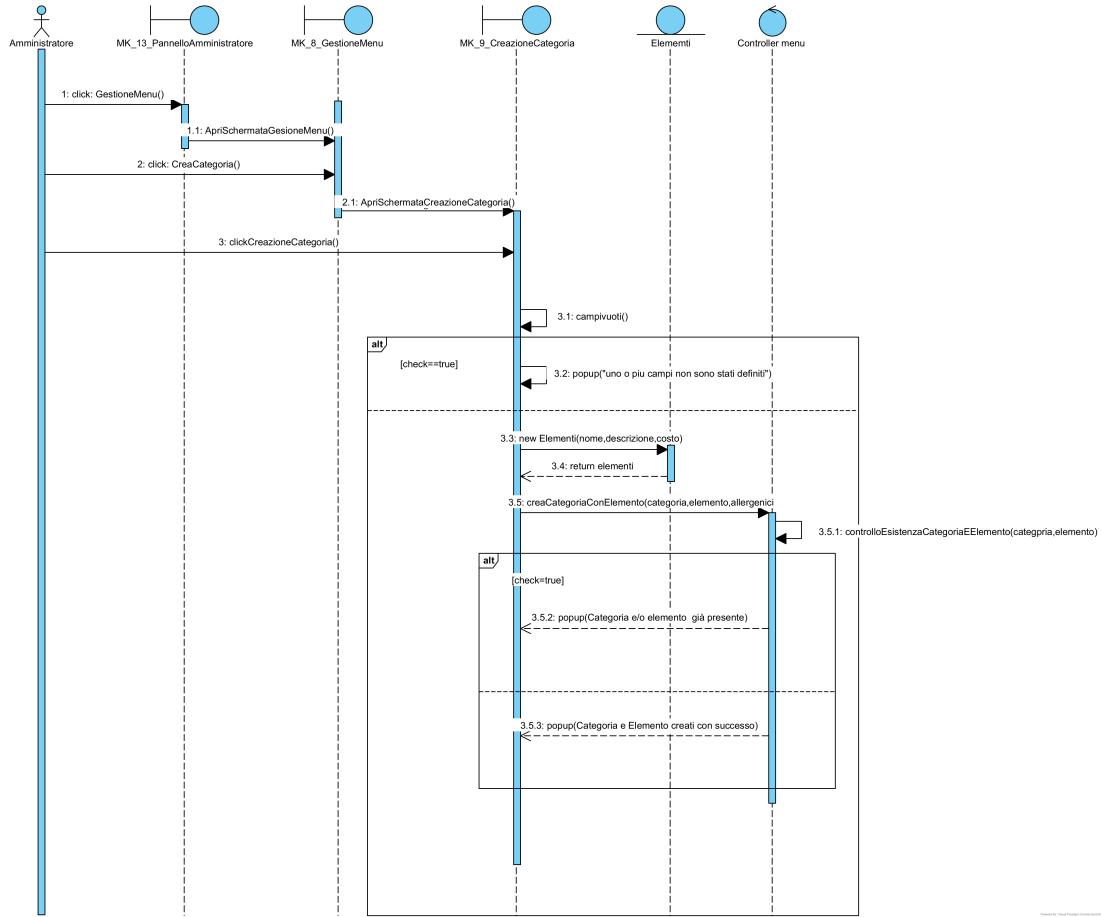


Figura 34: Sequence Diagram di analisi - Creazione Categoria

3.1.14 Sequence Diagram di analisi - Creazione Utenza

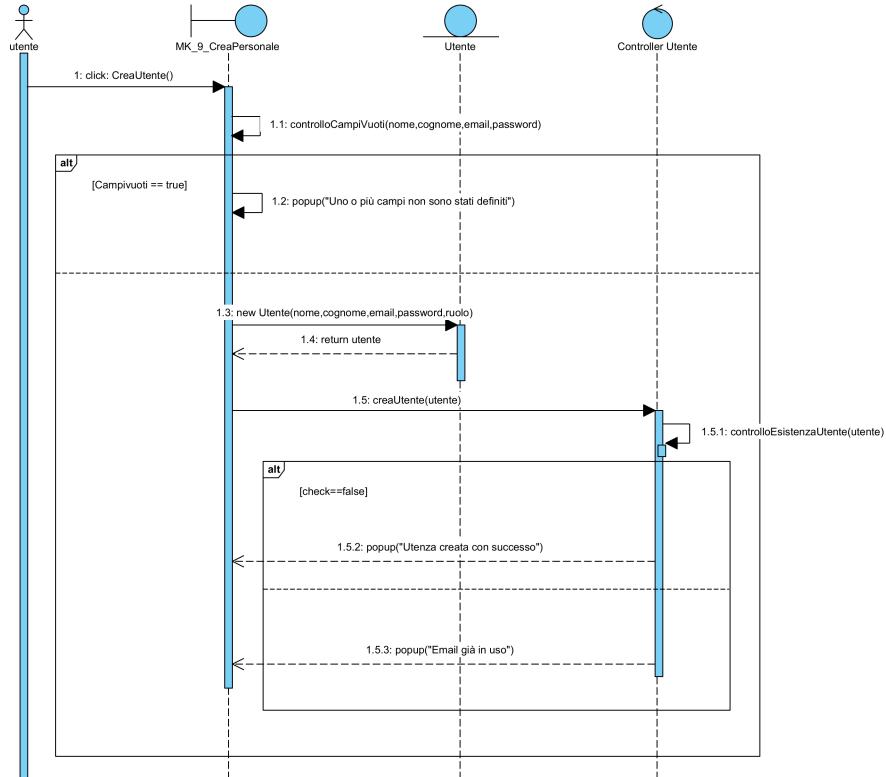


Figura 35: Sequence Diagram di analisi - Creazione utenza

3.2 Statechart di analisi

3.2.1 StateChart di analisi - Autenticazione

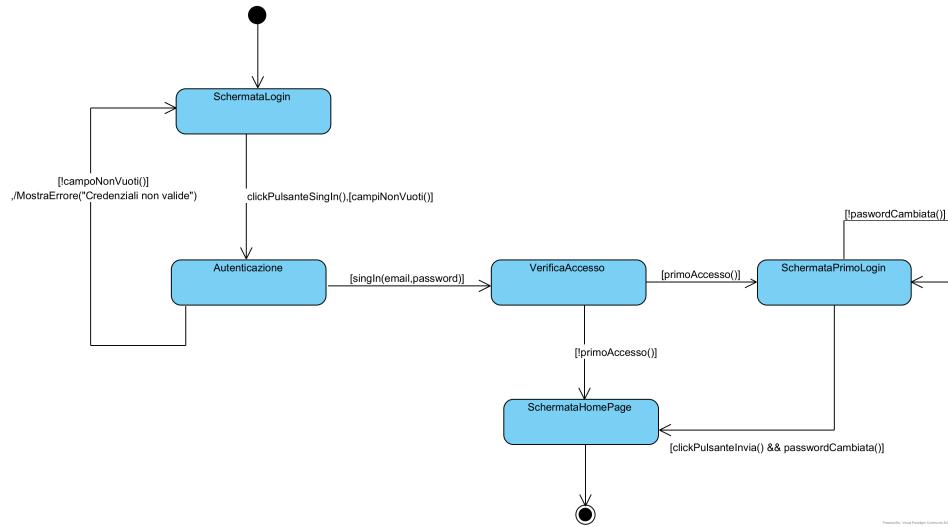


Figura 36: StateChart di analisi - Autenticazione

3.2.2 StateChart di analisi - Gestione Menu

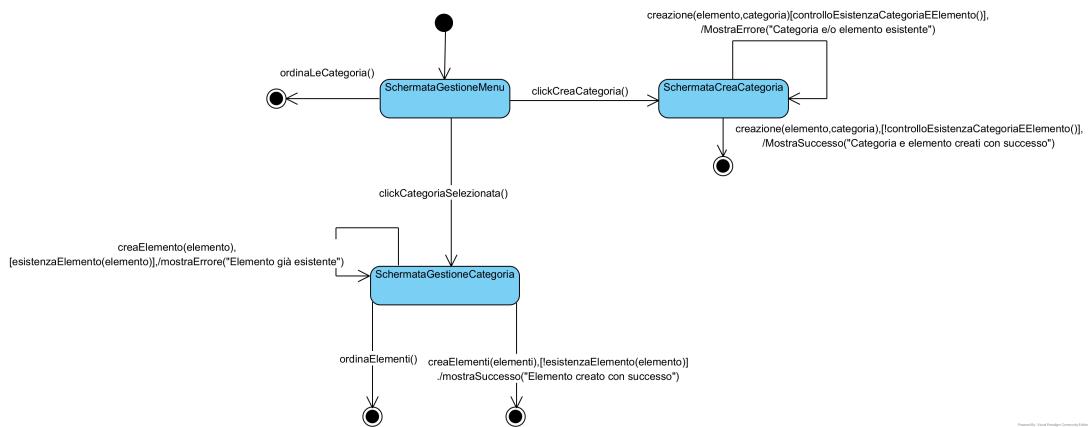


Figura 37: StateChart di analisi - Gestione Menu

3.2.3 StateChart di analisi - Visualizza Avviso

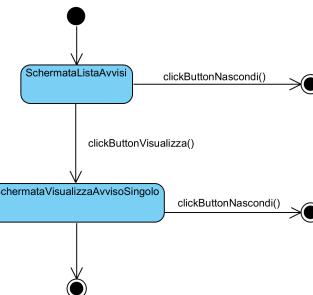


Figura 38: StateChart di analisi - Visualizza Avviso

3.2.4 StateChart di analisi - Creazione Avviso

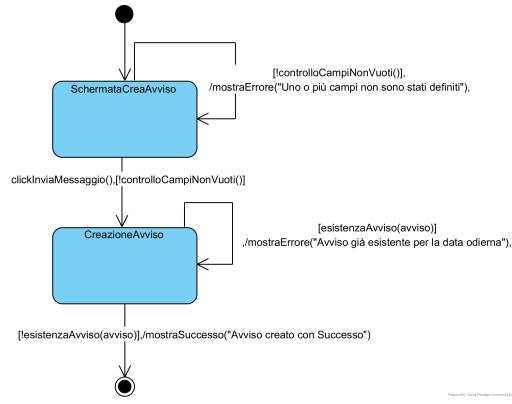


Figura 39: StateChart di analisi - Creazione Avviso

3.2.5 StateChart di analisi - Creazione Utenza

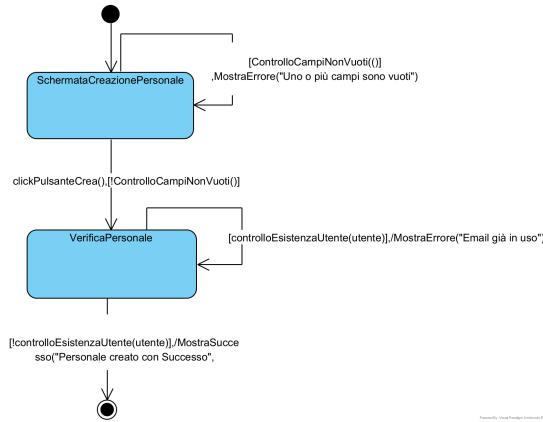


Figura 40: StateChart di analisi - Creazione utenza

3.2.6 StateChart di analisi - Gestione dell'ordine creato

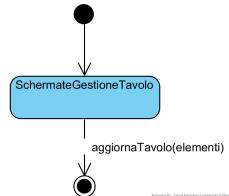


Figura 41: StateChart di analisi - Gestione dell'ordine creato

3.2.7 StateChart di analisi - Registrazione ordine

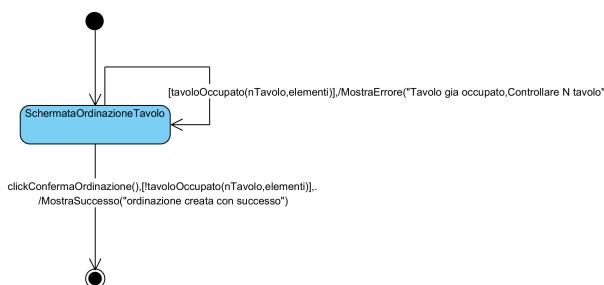


Figura 42: StateChart di analisi - Registrazione ordine

4 Design di Sistema

4.1 Analisi Architetturale

L'architettura utilizzata per la realizzazione del software "Ratatouille23" comprende un insieme di soluzioni e strategie basate su famosi **Design Pattern**. In particolare, è stato utilizzato l'architettura **client-server**, in cui i client sono dispositivi mobili basato su sistema operativo **Android** e il server backend viene utilizzato per gestire i servizi che offre ai client.

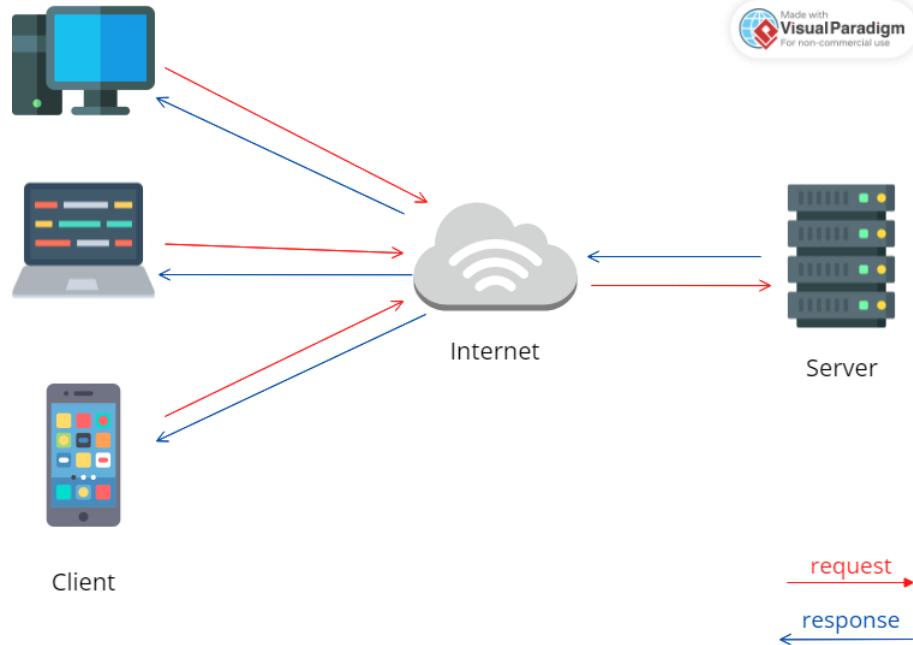


Figura 43: Architettura client-server

Il Client è stato sviluppato specificamente per i dispositivi mobili **Android**, in modo da consentire agli utenti di accedere al servizio tramite smartphone, tablet e computer basato su Android. Il Server è stato sviluppato in linguaggio **Java** che ci permette di gestire le comunicazioni del client tramite il protocollo HTTP, in modo da ricevere le richieste dai client e l'invio delle risposte con i dati richiesti e/o le conferme delle operazioni eseguite in modo asincrono.

Come già detto precedentemente la comunicazione avviene tramite protocollo HTTP dove il server backend espone delle API² che definisce il modo in cui i client interagiscono con il server backend(maggiori dettagli al capitolo 4.1.2).

Ricapitolando l'architettura client-server offre numerosi vantaggi come:

- Separazione delle responsabilità tra il client e il server, in cui il client si occupa dell'interfaccia utente, mentre il server gestisce la logica di business, l'accesso alle risorse richieste.
- Scalabilità del sistema che comprende la possibilità di aggiornare e modificare client e/o server in modo indipendente

²Application Programming Interface

4.1.1 Descrizione architettura Cloud

Ratatouille23 si propone di fornire un backend efficiente, sicuro e scalabile, in grado di adattarsi alle esigenze e alle variazioni di carico.

Per la realizzazione, la scelta è stata quella di utilizzare tecnologie all'avanguardia offerte dai servizi di public Cloud Computing di AWS³

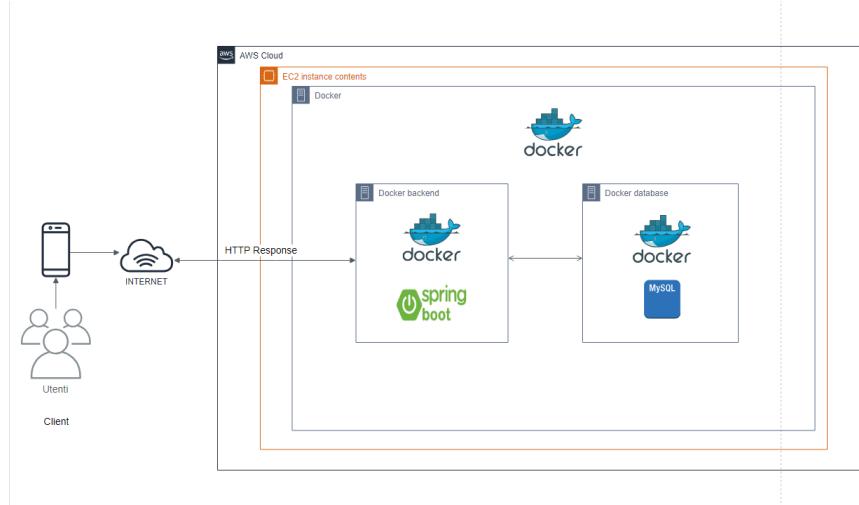


Figura 44: AWS

Le tecnologie utilizzate per tale realizzazione sono:

- **Springboot:** esso è un componente chiave per la realizzazione del backend. Springboot è un framework di sviluppo Java, grazie ad esso è stato possibile realizzare un insieme di API REST, consentendo una gestione semplificata delle risorse e delle operazioni CRUD⁴.
- **AWS EC2:** è un servizio di cloud computing scalabile ed affidabile. Con esso è possibile creare ed eseguire istanze di server virtuali in modo rapido e semplice, ci consente di ridimensionare le risorse in base alle nostre esigenze.
- **Docker:** è una piattaforma open-source, esso facilita e automatizza il processo di sviluppo e rilascio di applicazioni software tramite la creazione di contenitori leggeri e autonomi. Questi contenitori contengono tutto il necessario per eseguire l'applicazione, includendo il codice sorgente, le librerie e le dipendenze.

La scelta di distribuire sia il backend che il database in dei container Docker, perché permette moltissimi vantaggio, come:

- **Portabilità :** possibilità di essere eseguiti su qualsiasi piattaforma che possa far girare Docker
- **Scalabilità :** possibilità di aumentare e/o diminuire le risorse a loro allocate in modo semplice e veloce
- **Sicurezza :** i container sono isolati tra di loro e quindi se un container viene compromesso gli altri non saranno compromessi.

³<https://aws.amazon.com/it/>

⁴CRUD : C- create, R- read, U- Update, D- delete

4.1.2 Descrizione del server

Il Server di Ratatouille23 è stato scritto interamente in **JAVA** con l'utilizzo del framework **Spring-boot**, che grazie ad esso e alla sua semplicità di utilizzo è stato possibile esporre delle API REST. Alla fine di semplificare lo sviluppo e la gestione, è stato adottato il **design pattern MVC**.

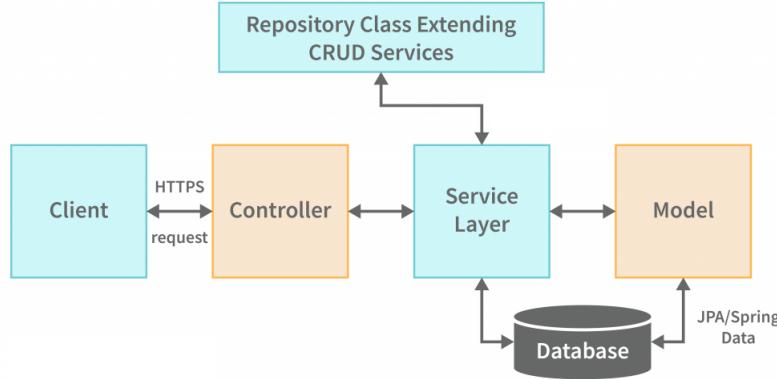


Figura 45: Pattern Architetturale del server

Questo pattern permette una chiara separazione delle responsabilità. Ha permesso di rendere il codice più modulare, mantenibile e riutilizzabile, permettendo anche un sviluppo semplice e veloce dell'API REST.

Il design pattern MVC segue delle linee guida che aiutano lo sviluppo di applicazione con Springboot, Esse sono:

- **Model:** i model rappresentano la struttura dei dati dell'applicazione, esse rappresentano i dati delle entità del nostro database.
- **Controller:** il Controller gestisce le richieste in arrivo dal client e gestisce la logica dell'applicazione. Esso gestisce tutti i metodi HTTP che sono GET,POST,PUT,DELETE.
- **Service:** Il Service è il layer intermedio tra il repository e il controller. Esso si occupa della logica di recupero dei dati dal repository, della logica di business e della manipolazione dei dati.
- **Repository:** I repository sono responsabili della comunicazione della logica con il database utilizzato. In Java vengono utilizzate le JPA⁵ per interagire con il database e eseguire operazioni CRUD.

4.1.3 Descrizione API REST

Di seguito sono riportate le API REST implementate e le relative route principali per Ratatouille23:

API REST MAIN ROUTE		
CAMPO	ROUTE	DESCRIZIONE
Utente	http://server-ip:port/api/account	Main route per le operazioni sugli utenti
Menu	http://server-ip:port/api/menu	Main route per le operazioni sul menu
Avviso	http://server-ip:port/api/avviso	Main route per le operazioni sugli avvisi
Tavolo	http://server-ip:port/api/tavolo	Main route per le operazioni sui tavoli

Tabella 5: Tabelle API REST MAIN ROUTE

Adesso verranno riportate le sotto-route dei vari campi.

⁵ Java Persistence API

API REST UTENTE ROUTE		
TIPO	FUNZIONE	DESCRIZIONE
PATCH	reImpostaPassword	Funzione che permette di reimpostare la password al primo accesso
POST	creaUtenza	Funzione che permette di creare utenza per il personale
	login	Funzione che permette di recuperare le informazioni di accesso e di generare il token
GET	infoUtente	Funzione che permette di ritornare le informazioni di un utente

API REST MENU ROUTE		
TIPO	FUNZIONE	DESCRIZIONE
GET	tutteLeCategorie	Funzione che ritorna tutte le categorie del Menu
	tuttiGliElementi	Funzione che ritorna tutti gli elementi di una determinata categoria
	autocompleteOpenFood	Funzione che ritorna il prodotto di APIFOOD cercato tramite nome
	cancellaElemento	Funzione che permette di cancellare un elemento
DELETE	creaCategoriaConElemento	Funzione che permette di creare una Categoria con il suo elemento
	aggiungiElemento	Funzione che permette di aggiungere un elemento ad una determinata Categoria
	ordinaElementi	Funzione che permette di ordinare gli elementi di una determinata Categoria
	ordinaCategorie	Funzione che permette di ordinare le categorie
POST	creaAvviso	Funzione che permette di creare un Avviso
	aggiornaStatoAvviso	Funzione che permette di aggiornare lo stato di un Avviso

API REST AVVISO ROUTE		
TIPO	FUNZIONE	DESCRIZIONE
GET	returnAvvisiDiUnoSpecificoUtente	Funzione che ritorna tutti gli avvisi che un utente ha visualizzato e non
POST	creaAvviso	Funzione che permette di creare un Avviso
PATCH	aggiornaStatoAvviso	Funzione che permette di aggiornare lo stato di un Avviso

Per maggiori informazioni è possibile scaricare la JavaDoc QUI

API REST TAVOLO ROUTE		
TIPO	FUNZIONE	DESCRIZIONE
GET	elementiDisponibili	Funzione che ritorna tutti gli elementi visibili
	elencoTavoliOccupati	Funzione che ritorna tutti i tavoli Occupati
	ritornoInformazioniDelTavolo	Funzione che ritorna le informazioni di un tavolo
PATCH	aggiornaElementiAlTavolo	Funzione che permette di aggiornare gli elementi ordinate di un Tavolo
POST	ordinazioneTavolo	Funzione che permette di creare un ordinazione di un Tavolo

4.1.4 Descrizione del Client

Il client di Ratatouille23 è stato sviluppato interamente in **Android**.

La decisione di sviluppare il client interamente in Android perché esso offre diversi vantaggi, tra cui:

- **Open-source** : essendo un sistema operativo Open-source è possibile cercare librerie create appositamente da programmatore e facilmente integrabili.
- **Vasta gamma di dispositivi**: Android è uno dei sistemi operativi più diffuso al mondo, con questo è possibile raggiungere una vasta gamma di utenti.
- **Strumenti di sviluppo**: Android offre una vasta di strumenti per lo sviluppo di Applicazioni per esempio Android Studio.

Il design pattern utilizzato per lo sviluppo è **MVP** (Model-View-Presenter)

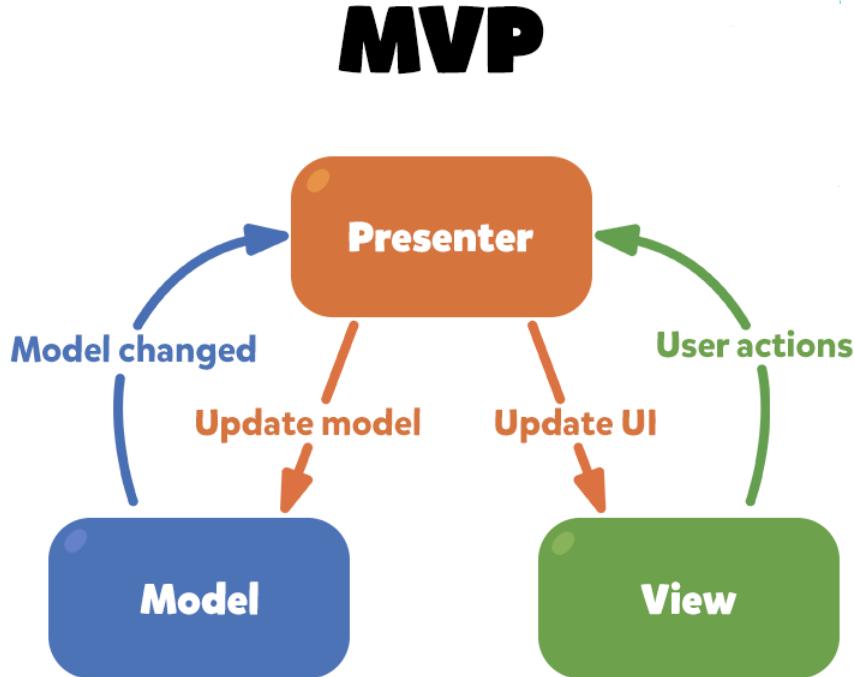


Figura 46: Design Pattern MVP

Come possiamo notare dall'immagine notiamo che il design pattern è composto da tre componenti:

- **Model**: rappresenta i dati ed è responsabile della logica di business e della comunicazione con il database. Il Model si occupa di accedere ai dati necessari e di applicare le regole di business definite.
- **View**: rappresenta l'interfaccia utente dell'applicazione Android. Si occupa di visualizzare i dati e di tenere traccia delle azioni effettuate dagli utenti

- **Presenter:** si occupa delle comunicazione tra la View e il Model. Esso si occupa di recuperare/aggiornare i dati dei model e applicare la logica dell’interfaccia utente in modo di aggiornare i dati visualizzati e degli elementi visualizzati

4.1.5 Classi di supporto

Per lo sviluppo di Android sono state utilizzate le seguenti librerie di supporto:

- **Retrofit**⁶: è una libreria per Android che semplifica la gestione delle richieste HTTP verso micro-servizi che utilizzano le chiamate API REST
- **RxJava**⁷: è una libreria per la programmazione basata sugli eventi, essa aiuta a semplificare le operazioni asincrone. Viene utilizzato per gestire le chiamate alle API con Retrofit in modo asincrono.

Per lo sviluppo del Back-end sono state utilizzate le seguenti librerie di supporto:

- **JWT:JSON WEB TOKEN**⁸: è uno standard che fornisce un metodo sicuro e compatto per la trasmissione di informazioni tra il client e server in formato **JSON**⁹. Esso è stato utilizzato per l’autenticazione e l’autorizzazione per le chiamate API REST. Le informazioni contenute nel Token, ci ha permesso di mettere in sicurezza le API in modo che solo gli utenti che hanno i ruoli necessari potessero accedere a quelle informazioni.
- **Lombok**¹⁰: è una libraria Java, il suo scopo è quello di semplificare lo sviluppo e di ridurre la quantità di codice ripetitivo come ad esempio i getter e setter delle classi.

⁶<https://square.github.io/retrofit/>

⁷<https://github.com/ReactiveX/RxJava>

⁸<https://jwt.io/>

⁹JavaScript Object Notation

¹⁰<https://projectlombok.org/setup/maven>

4.2 Diagramma delle classi di design

In questa sezione verranno riportati tutti i class diagram di design che sono stati individuati durante lo sviluppo dell'applicazione.

4.2.1 Class diagram di design - Login

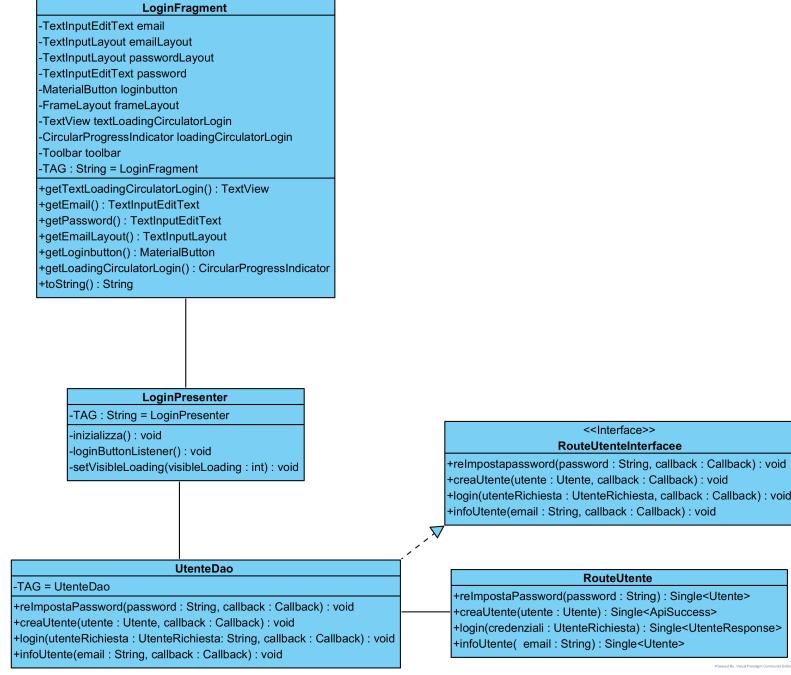


Figura 47: Class diagram di design - Login

4.2.2 Class diagram di design - Primo Accesso

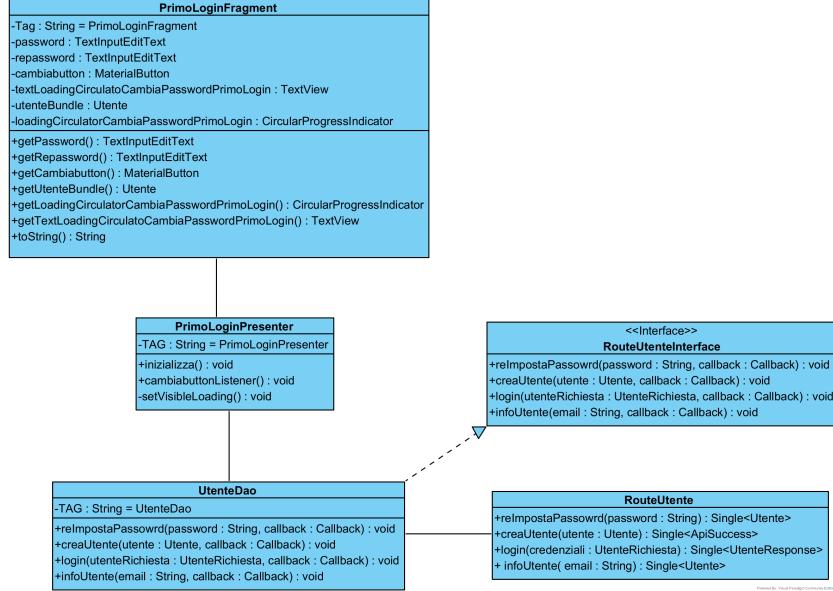


Figura 48: Class diagram di design - Primo Accesso

4.2.3 Class diagram di design - Creazione Utenza

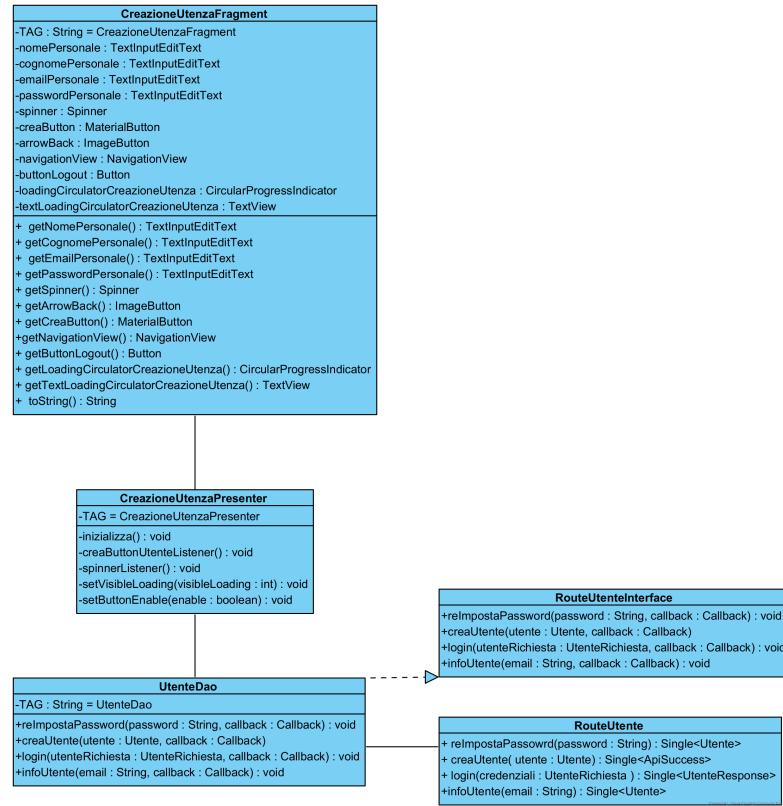


Figura 49: Class diagram di design - Creazione Utenza

4.2.4 Class diagram di design - Creazione Avviso

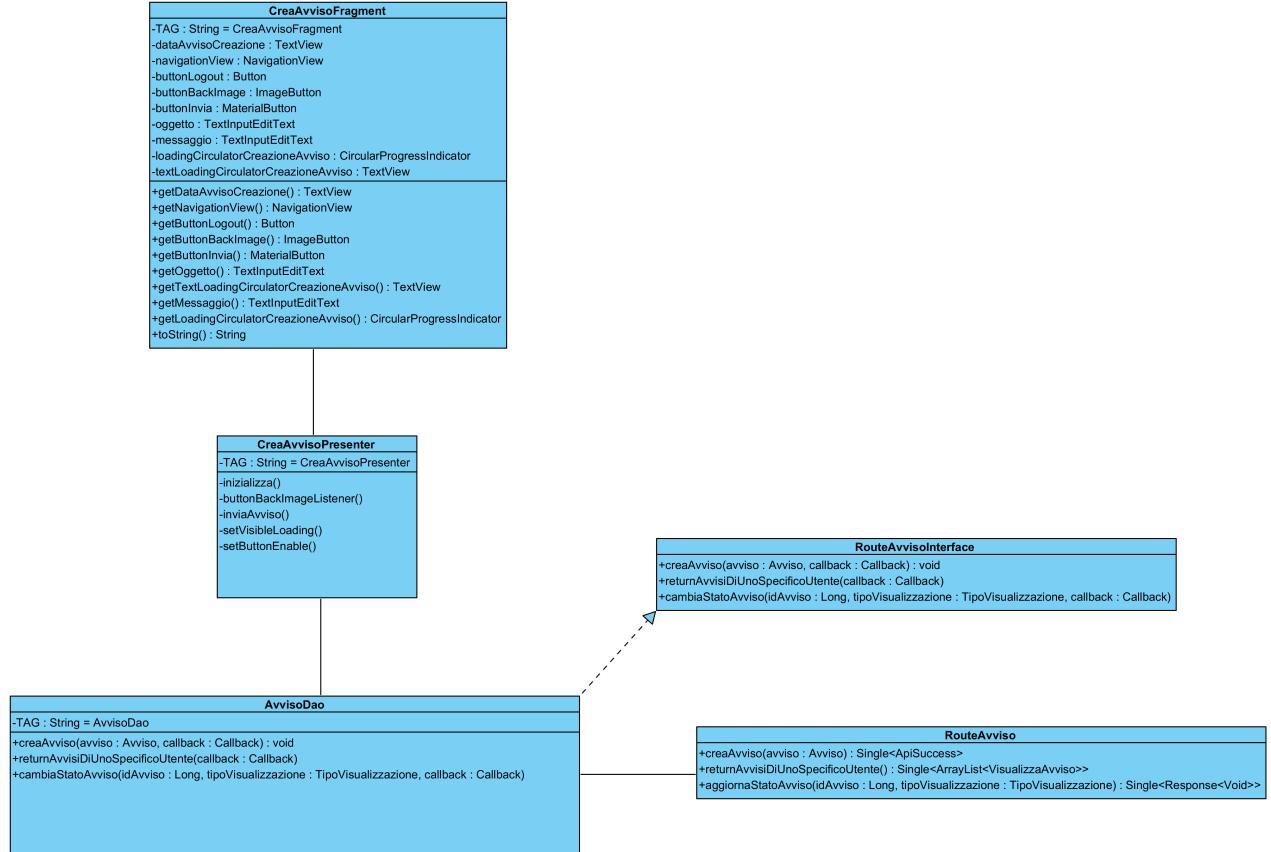


Figura 50: Class diagram di design - Creazione Avviso

4.2.5 Class diagram di design - Gestione Menù

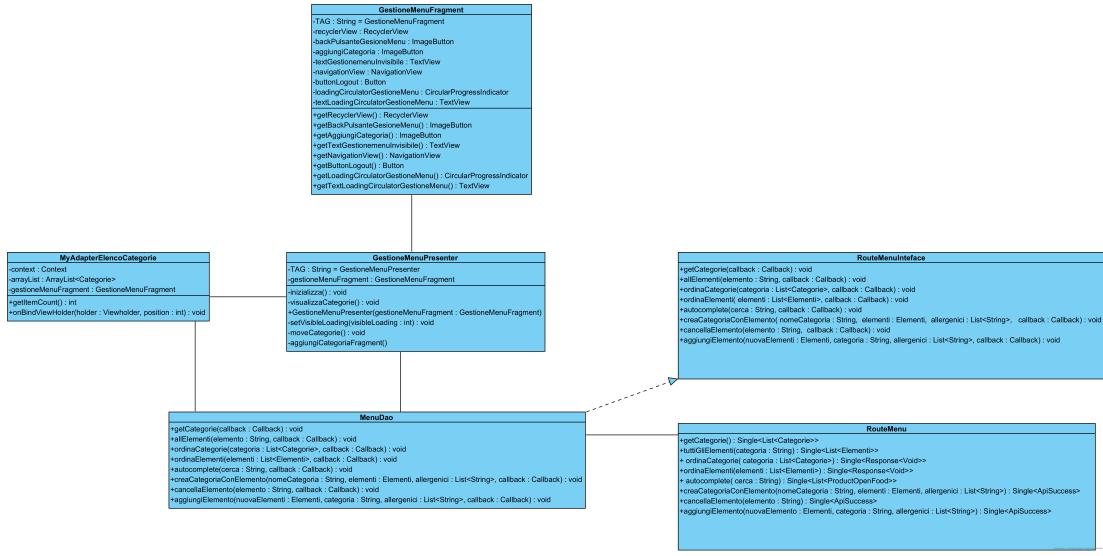


Figura 51: Class diagram di design - Gestione Menù

4.2.6 Class diagram di design - Gestione Categorie

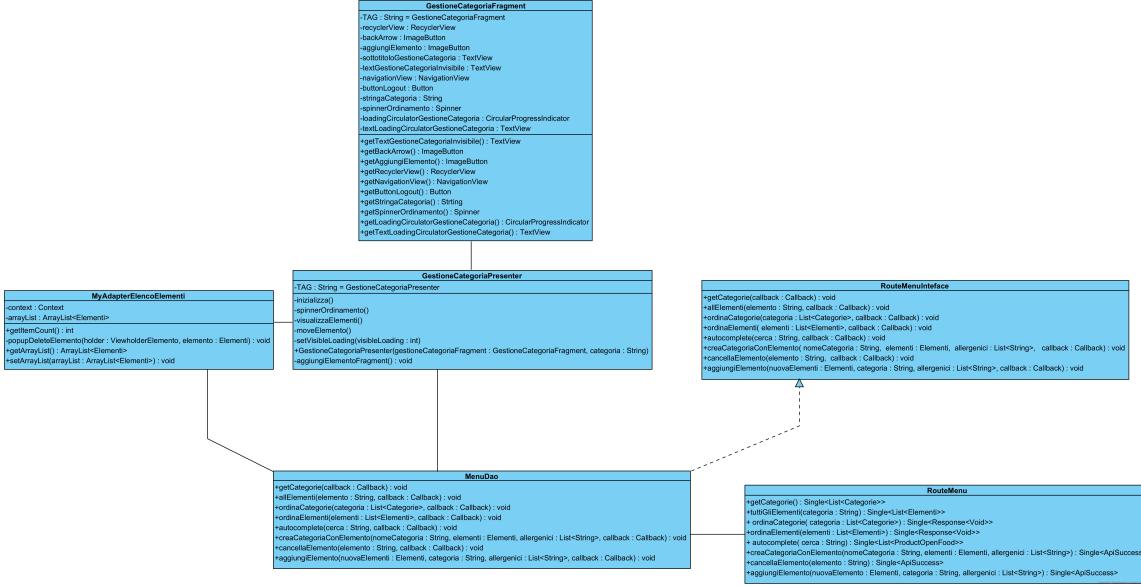


Figura 52: Class diagram di design - Gestione Categoria

4.2.7 Class diagram di design - Aggiungi Elemento alla Categoria

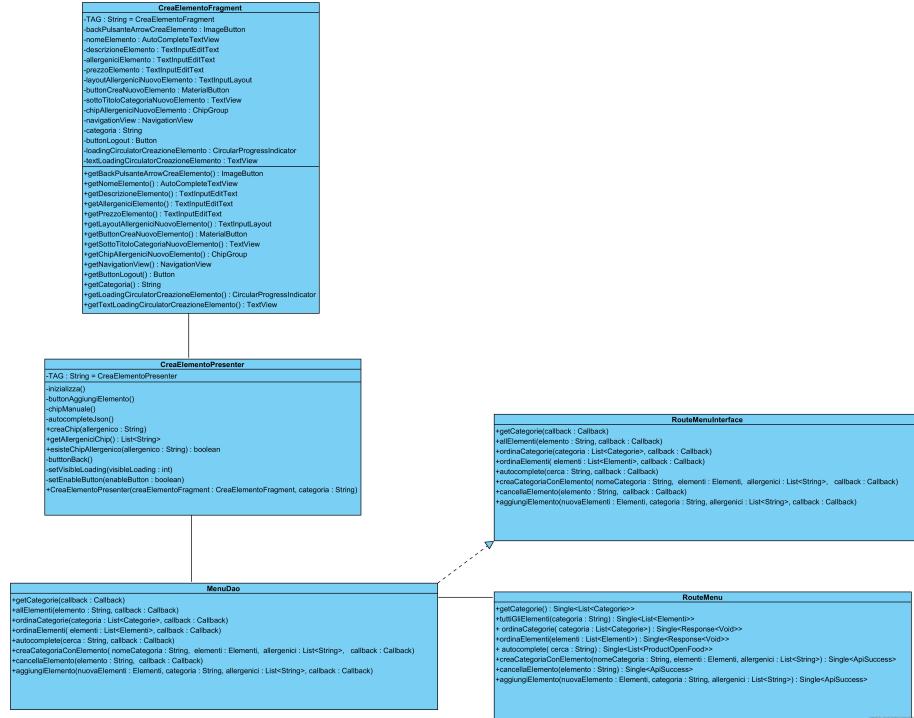


Figura 53: Class diagram di design - Aggiungi Elemento alla Categoria

4.2.8 Class diagram di design - Registrazione ordine

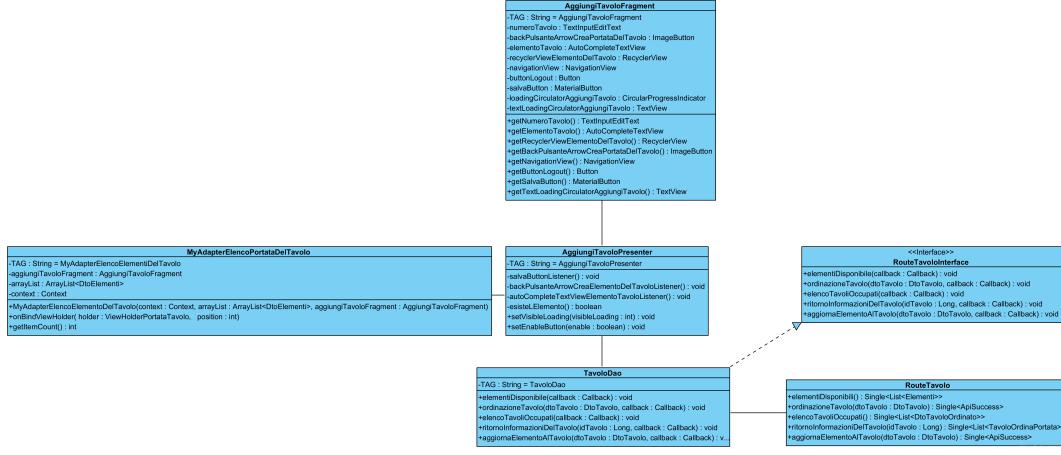


Figura 54: Class diagram di design - Registrazione ordine

4.3 Class diagram di design - Gestione dell'ordine creato

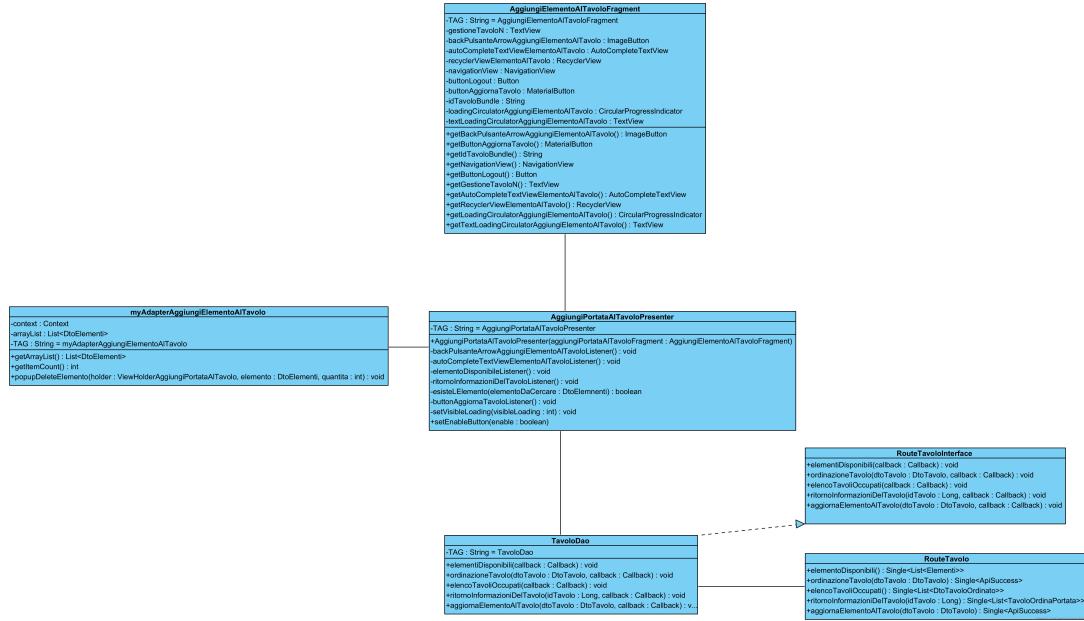


Figura 55: Class diagram di design - Gestione dell'ordine creato

4.4 Diagramma di stato di design

4.4.1 Creazione Avviso

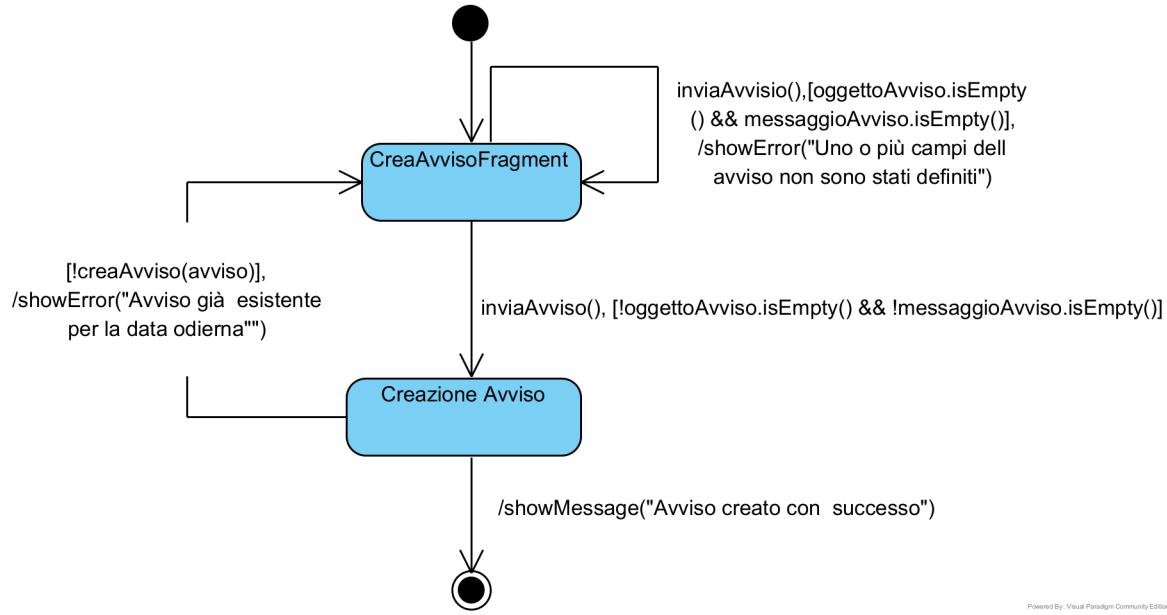


Figura 56: Diagramma di stato di design - Creazione Avviso

4.5 Diagramma di sequenza di design

4.5.1 Diagramma di sequenza di design - Creazione Utenza

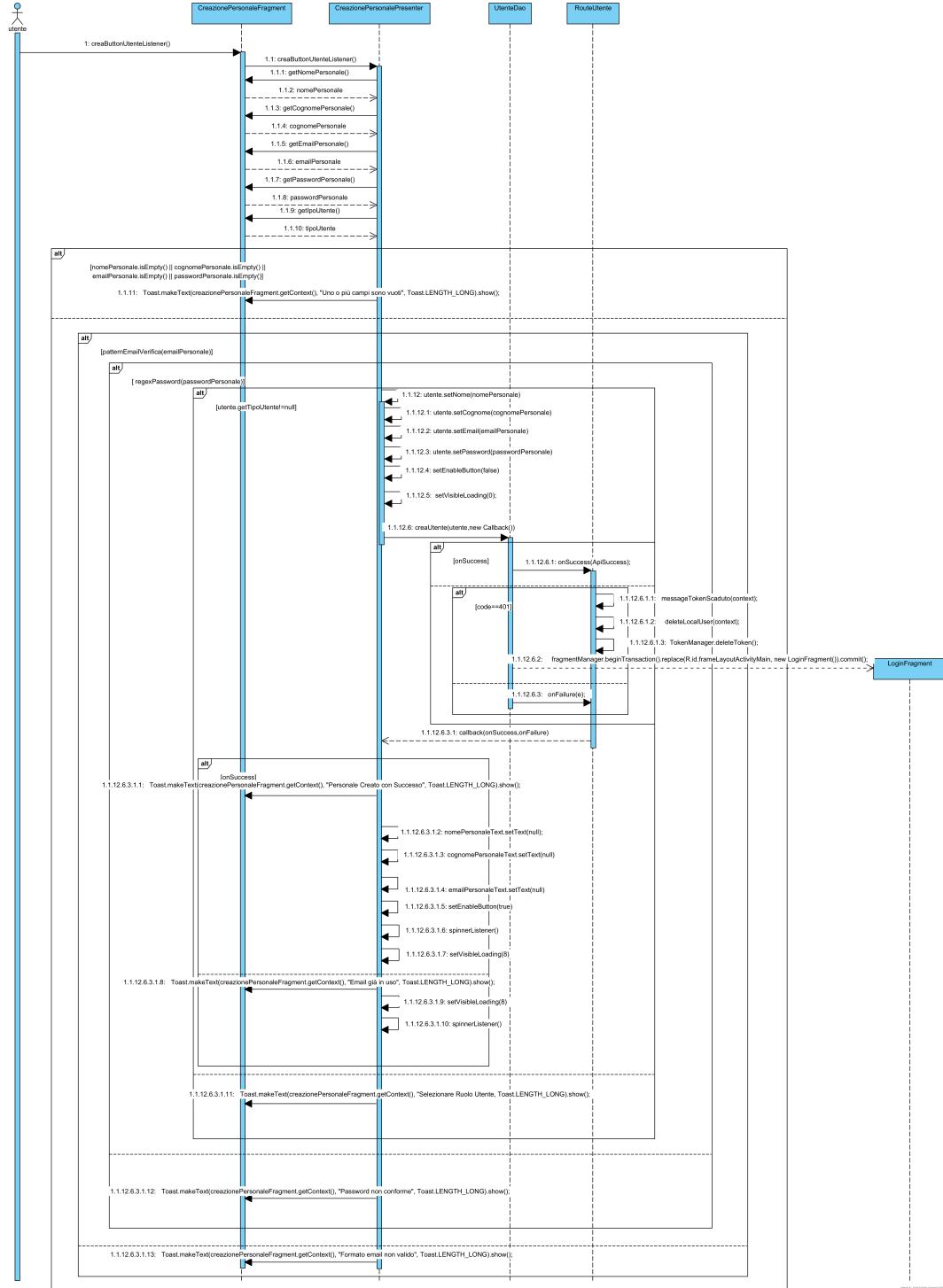


Figura 57: Diagramma di sequenza di design - Creazione Utenza

4.5.2 Diagramma di sequenza di design - Creazione Avviso

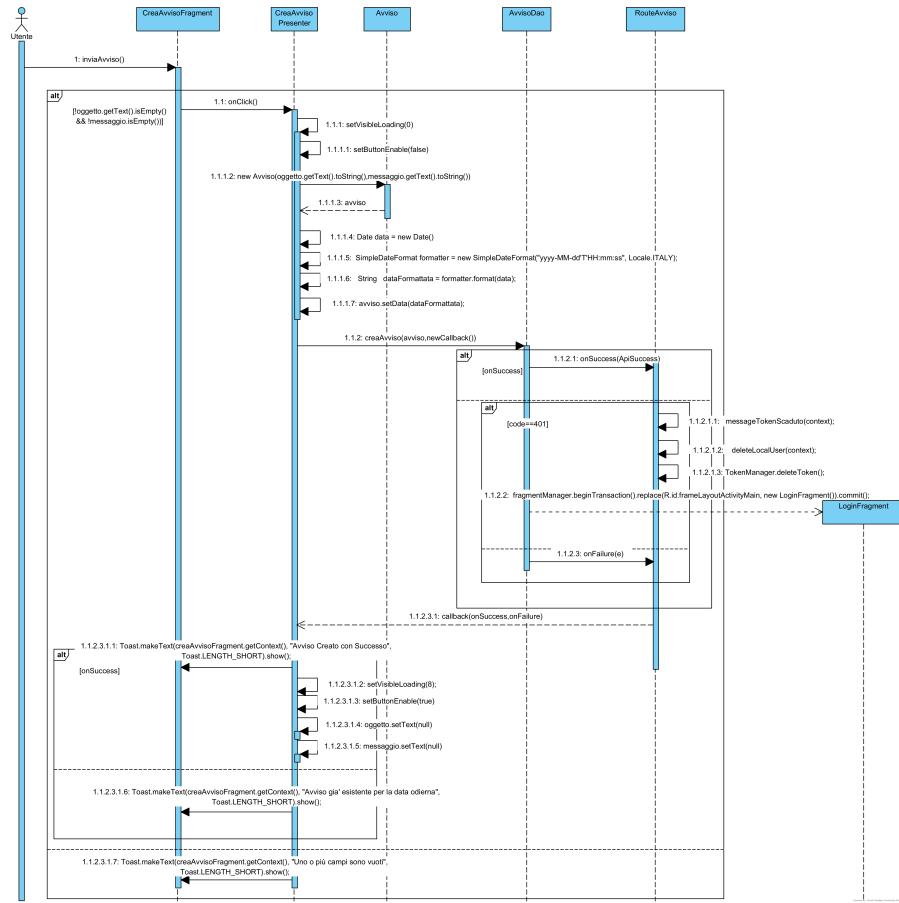


Figura 58: Diagramma di sequenza di design - Creazione Avviso

4.5.3 Diagramma di sequenza di design - Visualizza Avviso

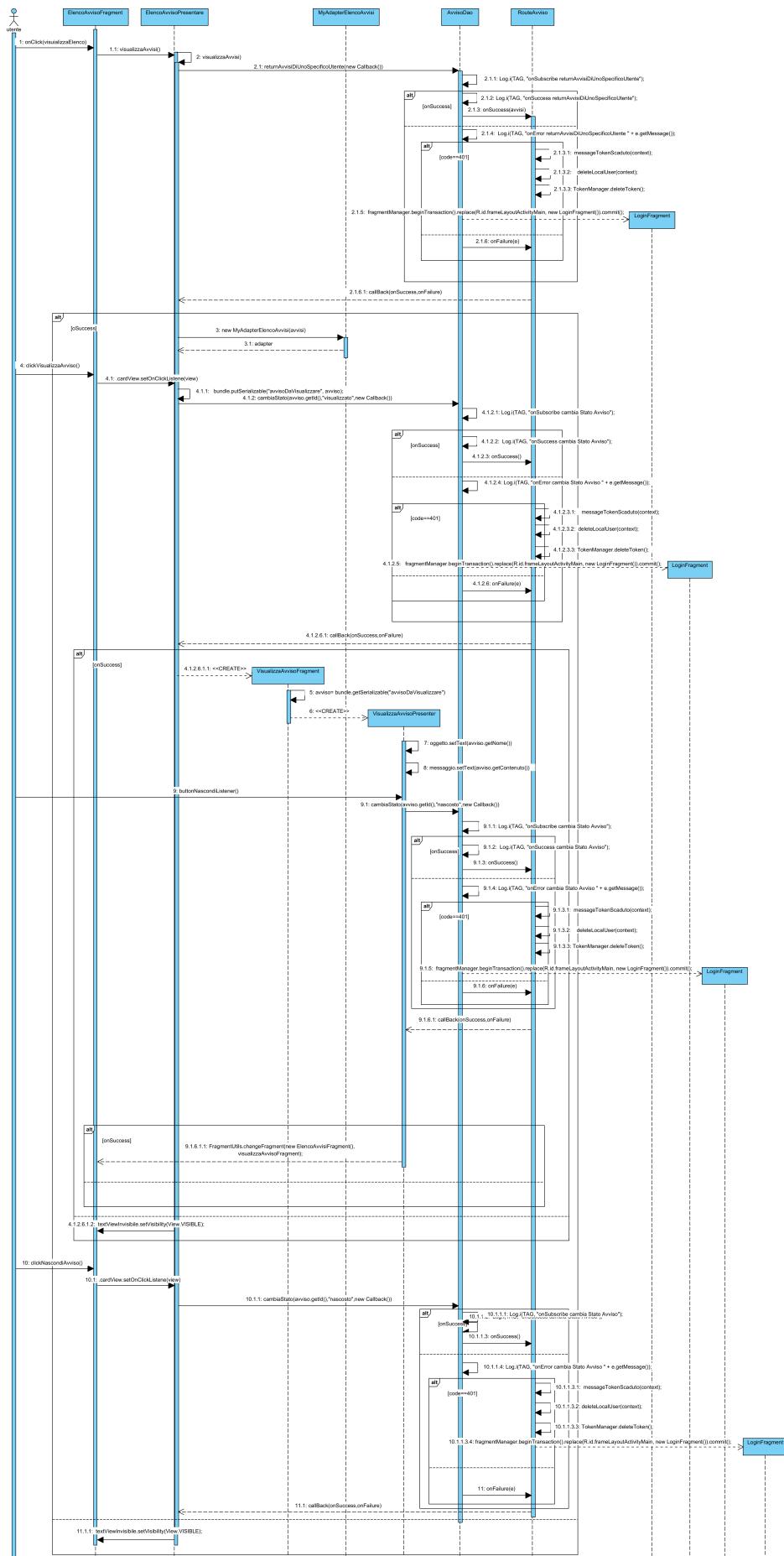


Figura 59: Diagramma di sequenza di design - Visualizza Avviso

4.5.4 Diagramma di sequenza di design - Registrazione Ordine

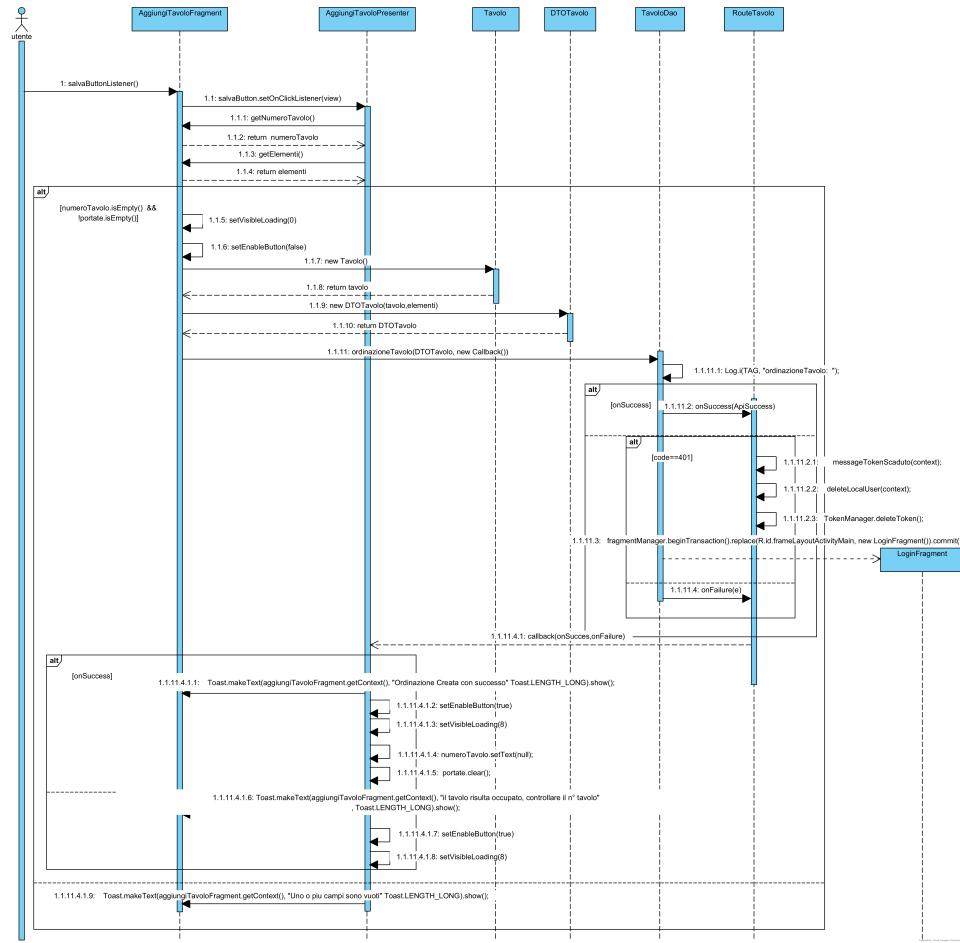


Figura 60: Diagramma di sequenza di design - Registrazione Ordine

5 Codice xUnit

In questa sezione verranno riportati i testing Junit di quattro metodi con almeno due parametri. Nella fase di testing sono stati utilizzati i seguenti strumenti:

- **JUnit**¹¹: è un framework open-source, viene usato per scrivere ed eseguire testing unitario su codice Java, offre una gamma di servizi utili a testare in modo accurato il codice.
- **Mockito**¹²: è un framework open-source per Java, Permette di creare oggetti finti nominati **mock** alla fine di simulare i componenti durante la fase di testing.

5.1 Testing BlackBox

In questa sezione verranno riportati i metodi testati in strategia BlackBox, essi sono:

- aggiungiElemento: Metodo che permette di aggiungere un nuovo elemento ad una categoria.
- creaAvviso: metodo che permette di creare un nuovo avviso.

5.1.1 Testing BlackBox aggiungiElemento

Adesso verrà riportata la firma del metodo da testare:

```
public ResponseEntity<ApiSuccess> aggiungiElemento(Elementi nuovaElemento,  
                                                 String categoria,  
                                                 Optional<List<String>> allergenici)  
                                                 throws BadRequestException
```

Per il testing del metodo sono state individuate le classi di equivalenza:

Per "**valore non valido**" intendiamo che il codice dovrebbe lanciare un'eccezione e quindi fermare l'esecuzione del metodo. Al contrario "**valore valido**" intendiamo che il valore inserito è valido e può portare a buon fine il metodo.

Il parametro **Elemento** essendo una classe sono stati individuati i seguenti classi di equivalenza.

Elementi nuovaElemento				
Nome N	Tipo Parametro	nome parametro	Insieme di valori	Effetto
N1	String	nuovaElemento	""	valore non valido
N2	String	nuovaElemento	"stringa piena"	valore valido
Nome D				
Nome D	Tipo Parametro	nome parametro	Insieme di valori	Effetto
D1	String	descrizione	""	valore non valido
D2	String	descrizione	"stringa piena"	valore valido
Nome P				
Nome P	Tipo Parametro	nome parametro	Insieme di valori	Effetto
P1	BigDecimal	prezzo	numero minore e/o uguale a zero	valore non valido
P2	BigDecimal	prezzo	numero maggiore strettamente di zero	valore valido

Per il parametro **categoria** sono state individuate le seguenti classi di equivalenza:

String categoria				
Nome C	Tipo Parametro	Nome Parametro	Insieme di valori	Effetto
C1	String	categoria	""	valore non valido
C2	String	categoria	"stringa piena"	valore valido

Per il parametro **allergenici** sono state individuate le seguenti classi di equivalenza:

Optional<List<String>>allergenici				
Nome C	Tipo Parametro	Nome Parametro	Insieme di valori	Effetto
A1	List<String>	allergenici	[] , null	valore valido
A2	List<String>	allergenici	["valore1",...]	valore valido

Per testare questo metodo è stato utilizzata la strategia SECT e sono stati individuati e implementati 32 metodi di test:

¹¹<https://junit.org/junit5/>

¹²<https://site.mockito.org/>

```

@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class AggiungiElementiBlackBox {
    @InjectMocks
    private ElementiService elementiService;
    @Mock
    private ElementiRepository elementiRepository;
    @Mock
    private CategorieService categorieService;
    @Mock
    private AllergeniciService allergeniciService;

    private Elementi elementi;
    private List<String> allergeniciList;
    private String categoria;

    @BeforeEach
    public void init() {
        elementi = new Elementi();
        allergeniciList = new ArrayList<>();
        allergeniciList.add("Allergenico1");
        allergeniciList.add("Allergenico2");

    }

    @Test//
    public void testAggiungiElemento_N1_D1_P1_C1_A1() {
        elementi.setNomeElemento("");
        elementi.setDescrizione("");
        elementi.setPrezzo(BigDecimal.valueOf(-150));
        allergeniciList = new ArrayList<>();
        categoria = "";
        assertThrows(BadRequestException.class , () -> {
            elementiService.aggiungiElemento(elementi, categoria, Optional.of(
                allergeniciList));
        });
    };
}

@Test
public void testAggiungiElemento_N1_D1_P1_C1_A2() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    categoria = "";

    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    });
};

@Test
public void testAggiungiElemento_N1_D1_P1_C2_A1() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    allergeniciList = new ArrayList<>();
    categoria = "primi";
    assertThrows(BadRequestException.class , () -> {

```

```

        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
}

@Test
public void testAggiungiElemento_N1_D1_P1_C2_A2() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    categoria = "primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D1_P2_C1_A1(){
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(1));
    categoria="";
    allergeniciList = new ArrayList<>();
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D1_P2_C1_A2(){
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(1));
    categoria="";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D1_P2_C2_A1(){
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(1));
    categoria="primi";
    allergeniciList = new ArrayList<>();
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D1_P2_C2_A2(){
    elementi.setNomeElemento("");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(1));
}

```

```

        categoria="primi";
        assertThrows(BadRequestException.class , () -> {
            elementiService.aggiungiElemento(elementi, categoria, Optional.of(
                allergeniciList));
        }
    );
}

@Test
public void testAggiungiElemento_N1_D2_P1_C1_A1() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    allergeniciList = new ArrayList<>();
    categoria = "";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D2_P1_C1_A2() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    categoria = "";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D2_P1_C2_A1() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.ZERO);
    categoria="primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria,null);
    }
);
}

@Test
public void testAggiungiElemento_N1_D2_P1_C2_A2() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(-1));
    categoria="primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria,Optional.of(
            allergeniciList));
    }
);
}

```

```

    @Test
    public void testAggiungiElemento_N1_D2_P2_C1_A1() {
        elementi.setNomeElemento("");
        elementi.setDescrizione("descrizione");
        elementi.setPrezzo(BigDecimal.valueOf(1));
        categoria="";
        allergeniciList=new ArrayList<>();
        assertThrows(BadRequestException.class , () -> {
            elementiService.aggiungiElemento(elementi, categoria,Optional.of(
                allergeniciList));
        }
    );
}

@Test
public void testAggiungiElemento_N1_D2_P2_C1_A2() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(1));
    categoria="";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria,Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N1_D2_P2_C2_A1() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(1));
    categoria="primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria,null);
    }
);
}

@Test
public void testAggiungiElemento_N1_D2_P2_C2_A2() {
    elementi.setNomeElemento("");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(5));
    categoria = "primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D1_P1_C1_A1() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    allergeniciList = new ArrayList<>();
    categoria = "";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

```

```

        }
    );
}

@Test
public void testAggiungiElemento_N2_D1_P1_C1_A2() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    allergeniciList = new ArrayList<>();
    categoria = "";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D1_P1_C2_A1(){
    elementi.setNomeElemento("elemento");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.ZERO);
    categoria="primi";
    allergeniciList=new ArrayList<>();
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D1_P1_C2_A2() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    categoria = "primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D1_P2_C1_A1() {
    elementi.setNomeElemento("elemnto");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.TEN);
    categoria="";
    allergeniciList=new ArrayList<>();
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D1_P2_C1_A2() {
    elementi.setNomeElemento("elemnto");
    elementi.setDescrizione("");
}

```

```

elementi.setPrezzo(BigDecimal.TEN);
categoria="";
assertThrows(BadRequestException.class , () -> {
    elementiService.aggiungiElemento(elementi, categoria, Optional.of(
        allergeniciList));
}
);
}
@Test
public void testAggiungiElemento_N2_D1_P2_C2_A1() {
    elementi.setNomeElemento("elemnto");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.TEN);
    categoria="primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, null);
    }
);
}

@Test
public void testAggiungiElemento_N2_D1_P2_C2_A2() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("");
    elementi.setPrezzo(BigDecimal.valueOf(5));
    categoria = "primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D2_P1_C1_A1() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    allergeniciList = new ArrayList<>();
    categoria = "";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D2_P1_C1_A2() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    categoria = "";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

```

```

@Test
public void testAggiungiElemento_N2_D2_P1_C2_A1(){
    elementi.setNomeElemento("elemento");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.ZERO);
    categoria="primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria,null);
    }
);
}

@Test
public void testAggiungiElemento_N2_D2_P1_C2_A2() {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(-150));
    categoria = "primi";
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D2_P2_C1_A1(){
    elementi.setNomeElemento("elemento");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.TEN);
    categoria="";
    allergeniciList=new ArrayList<>();
    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D2_P2_C1_A2(){
    elementi.setNomeElemento("elemento");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.TEN);
    categoria="";

    assertThrows(BadRequestException.class , () -> {
        elementiService.aggiungiElemento(elementi, categoria, Optional.of(
            allergeniciList));
    }
);
}

@Test
public void testAggiungiElemento_N2_D2_P2_C2_A2() throws BadRequestException {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(5));
    categoria = "primi";
    ResponseEntity<ApiSuccess> response = elementiService.aggiungiElemento(
        elementi, categoria, Optional.of(allergeniciList));
}

```

```

        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType());
        assertEquals("Elemento aggiunto con successo", response.getBody().getMessage());
    }

}

@Test
public void testAggiungiElemento_N2_D2_P2_C2_A1() throws BadRequestException {
    elementi.setNomeElemento("elementi");
    elementi.setDescrizione("descrizione");
    elementi.setPrezzo(BigDecimal.valueOf(5));
    categoria = "primi";
    allergeniciList=new ArrayList<>();
    ResponseEntity<ApiSuccess> response = elementiService.aggiungiElemento(
        elementi, categoria, Optional.of(allergeniciList));
    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType());
    assertEquals("Elemento aggiunto con successo", response.getBody().getMessage());
}

}

```

5.1.2 Testing BlackBox creaAvviso

Adesso verrà riportato la firma del metodo da testare:

```
public ResponseEntity<ApiSuccess> creaAvviso(Avviso avviso, String email) throws
    BadRequestException, NotFoundException
```

Per il testing del metodo sono state individuate le seguenti classi di equivalenza:

Per "**valore non valido**" intendiamo che il codice dovrebbe lanciare un'eccezione e quindi fermare l'esecuzione del metodo. Al contrario "**valore valido**" intendiamo che il valore inserito è valido e può portare a buon fine il metodo.

Il parametro **Avviso** essendo una classe sono state individuate le seguenti classi di equivalenza

Avviso avviso				
Nome O	Tipo Parametro	Nome Parametro	Insieme di valori	Effetto
O1	String	oggetto	""	valore non valido
O2	Sring	oggetto	"stringa piena"	valore valido
Nome C	Tipo Parametro	Nome Parametro	Insieme di valori	Effetto
C1	String	contenuto	""	valore non valido
C2	String	contenuto	"Stringa piena"	valore valido
Nome D	Tipo Parametro	Nome Parametro	Insieme di valori	Effetto
D1	Date	data	null	valore non valido
D2	Date	data	qualsiasi data	valore valido

Per il parametro **email** sono individuate le seguenti classi di equivalenza:

String email				
Nome E	Tipo Parametro	Nome Parametro	Insieme di valori	Effetto
E1	String	email	""	valore non valido
E2	String	email	pattern corretto	valore valido
E3	String	email	pattern sbagliato	valore non valido

Per testare questo metodo è stato utilizzato la strategia WECT e sono state individuate e implementate 18 metodi di test:

```

@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class CreaAvvisoBlackBox {

    @InjectMocks
    private AvvisoService avvisoService;

    @Mock
    private AvvisoRepository repository;

    @Mock
    private UtenteService utenteService;
    @Mock
    private VisualizzaAvvisoService visualizzaAvvisoService;

    @Mock
    private UtenteRepository utenteRepository;

    private Utente utente = new Utente();
    private String email;
    private Avviso avviso = new Avviso();

    @BeforeEach
    public void init() {

        utente.setNome("Ciro");
        utente.setCognome("de Cristofaro");
        utente.setEmail("prova@gmail.com");
        utente.setTipoUtente(TipoUtente.Amministratore);
    }

    @Test
    public void test_CreaAvviso_01_C1_D1_E1() {
        avviso.setOggetto("");
        avviso.setContenuto("");
        avviso.setData(null);
        email = "";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C1_D1_E3() {
        avviso.setOggetto("");
        avviso.setContenuto("");
        avviso.setData(null);
        email = "emailsbagliata.com@";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C1_D1_E2() {
        avviso.setOggetto("");
        avviso.setContenuto("");
        avviso.setData(null);
        email = "prova@gmail.com";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }
}

```

```

    @Test
    public void test_CreaAvviso_01_C1_D2_E1() {
        avviso.setOggetto("");
        avviso.setContenuto("");
        avviso.setData(new Date());
        email = "";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C1_D2_E2() {
        avviso.setOggetto("");
        avviso.setContenuto("");
        avviso.setData(new Date());
        email = "prova@gmail.com";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C2_D1_E1() {
        avviso.setOggetto("");
        avviso.setContenuto("Contenuto");
        avviso.setData(null);
        email = "";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C2_D1_E2() {
        avviso.setOggetto("");
        avviso.setContenuto("Contenuto");
        avviso.setData(null);
        email = "prova@gmail.com";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C2_D2_E1() {
        avviso.setOggetto("");
        avviso.setContenuto("Contenuto");
        avviso.setData(new Date());
        email = "";
        assertThrows(BadRequestException.class, () -> {
            avvisoService.creaAvviso(avviso, email);
        });
    }

    @Test
    public void test_CreaAvviso_01_C2_D2_E2() {
        avviso.setOggetto("");
        avviso.setContenuto("Contenuto");
        avviso.setData(new Date());

```

```

email = "prova@gmail.com";
assertThrows(BadRequestException.class, () -> {
    avvisoService.creaAvviso(avviso, email);
});
}

@Test
public void test_CreaAvviso_02_C1_D1_E1() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("");
    avviso.setData(null);
    email = "";
    assertThrows( BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C1_D1_E2() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("");
    avviso.setData(null);
    email = "prova@gmail.com";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C1_D2_E1() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("");
    avviso.setData(new Date());
    email = "";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C1_D2_E2() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("");
    avviso.setData(new Date());
    email = "prova@gmail.com";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C2_D1_E1() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("contenuto");
    avviso.setData(null);
    email = "";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test

```

```

public void test_CreaAvviso_02_C2_D1_E2() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("contenuto");
    avviso.setData(null);
    email = "prova@gmail.com";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C2_D2_E1() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("contenuto");
    avviso.setData(new Date());
    email = "";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C2_D2_E3() {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("contenuto");
    avviso.setData(new Date());
    email = "E-mailsbliat.com1";
    assertThrows(BadRequestException.class, () -> {
        avvisoService.creaAvviso(avviso, email);
    });
}

@Test
public void test_CreaAvviso_02_C2_D2_E2() throws InternalServerErrorException,
    BadRequestException, NotFoundException {
    avviso.setOggetto("oggetto");
    avviso.setContenuto("contenuto");
    avviso.setData(new Date());
    email = "prova@gmail.com";
    ResponseEntity<ApiSuccess> response = avvisoService.creaAvviso(avviso, email);
    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType());
    assertEquals("Avviso Creato con Successo", response.getBody().getMessage());
}
}

```

5.2 Testing WhiteBox

In questa sezione verranno riportati i metodi testati in strategia WhiteBox, essi sono:

- creaCategoriaConElemento: questo metodo permette di creare una categoria con un elemento
- reImpostaPassword: questo metodo permette di cambiare la password al primo accesso all'applicazione.

5.2.1 Testing WhiteBox creaCategoriaConElemento

Adesso verrà riportato metodo da testare :

```
public ResponseEntity<ApiSuccess> creaCategoriaConElemento(String nomeCategoria,
                                                               Elementi elemento,
                                                               Optional<List<String>>
                                                               allergenici) throws
                                                               BadRequestException,
                                                               NotFoundException {
    if (nomeCategoria == null || nomeCategoria.trim().isEmpty()) {
        throw new BadRequestException("Nome categoria non specificato", new
                                       Throwable("Nome categoria non specificato"));
    }

    if (existsByNomeCategoriaIgnoreCase(nomeCategoria)) {
        throw new BadRequestException("Categoria già presente", new Throwable("Categoria già presente"));
    }

    if (elementiService.controlloCampiElementoValidati(elemento)) {
        throw new BadRequestException("Uno o più campi della elemento non sono
                                      stati definiti", new Throwable("Uno o più campi della elemento non sono
                                      stati definiti"));
    }

    if (elementiService.checkIfElementoExists(elemento.getNomeElemento())) {
        throw new BadRequestException("Elemento già esistente", new Throwable("Elemento già esistente"));
    }

    if (elemento.getPrezzo().compareTo(BigDecimal.ZERO) <= 0) {
        throw new BadRequestException("Il prezzo deve essere maggiore di zero", new
                                       Throwable("Il prezzo deve essere maggiore di zero"));
    }

    Menu menu = menuService.findByNomeMenu(nomeMenu);

    Categorie nuovaCategoria = new Categorie();
    nuovaCategoria.setNomeCategoria(nomeCategoria);
    nuovaCategoria.setMenu(menu);
    List<Categorie> elementoOrdine = findAll();
    log.info("ord : {}", elementoOrdine);
    if (elementoOrdine.size() != 0) {
        int ordine = findByOrderByOrdineDesc().get(0).getOrdine();
        log.info("ordine : {}", ordine);
        nuovaCategoria.setOrdine(++ordine);
    }
    save(nuovaCategoria);

    if (allergenici != null && !allergenici.isEmpty()) {
        List<Allergenici> allergeniciList = new ArrayList<>();
        for (String allergenico : allergenici.get()) {
            Allergenici allergenicoOpt = allergeniciService.
                findOneByNomeAllergenicoIgnoreCase(allergenico);
            if (allergenicoOpt != null) {
```

```

        log.info("Allergenico già presente :{}", allergenicoOpt.
                  getNomeAllergenico());
        allergeniciList.add(allergenicoOpt);
    } else {
        log.info("Allergenico nuovo :{}", allergenico);
        Allergenici nuovoAllergenico = new Allergenici();
        nuovoAllergenico.setNomeAllergenico(allergenico);
        allergeniciService.save(nuovoAllergenico);
        allergeniciList.add(nuovoAllergenico);
    }

}
elemento.setAllergenici(allergeniciList);
}
Categorie categoria = findByNomeCategoria(nomeCategoria);
elemento.setCategoria(categoria);
elementiService.save(elemento);
log.info("categoria con elemento creata con successo");

return ResponseEntity.status(HttpStatus.CREATED)
    .contentType(MediaType.APPLICATION_JSON)
    .body(new ApiSuccess(201, "Categoria e Elemento creati con successo"));
}

```

Alla fine di testare il metodo è stato disegnato il suo grafo del flusso di controllo:

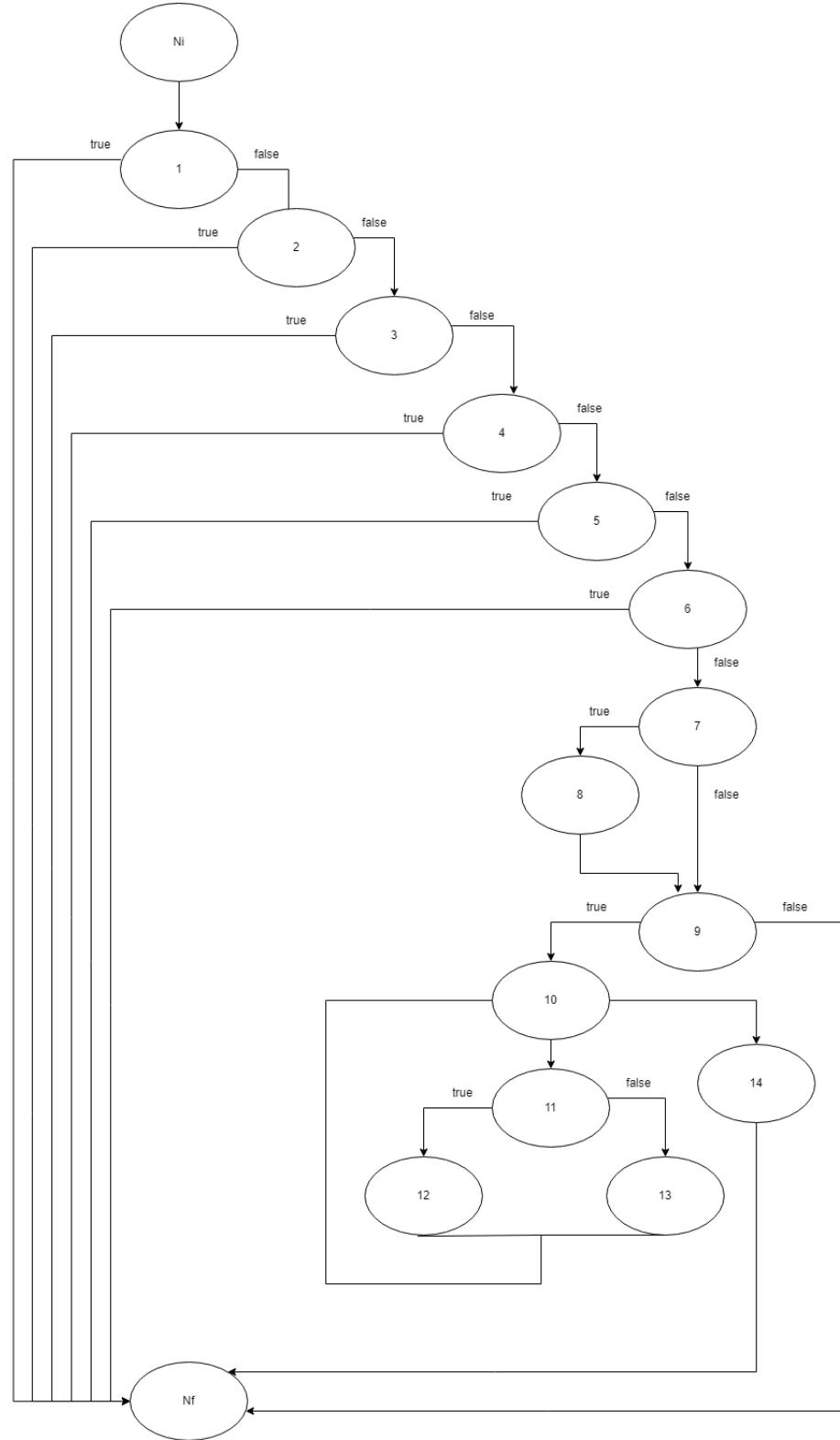


Figura 61: CFG - creaCategoriaConElemento

Dal Control Flow Graph ricaviamo i seguenti metodi al fine di ottenere una **Branch Coverage**:

- test_CreaCategoriaConElemento_WhiteBox_Path_1
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_9
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_9
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_9_10_11_12_14

- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_9_10_11_13_14
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_10_11_12_14
- test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_10_11_13_14

```

@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class CreaCategoriaConElementoWhiteBox {

    @InjectMocks
    private CategorieService categorieService;
    @Mock
    private ElementiService elementiService;
    @Mock
    private MenuService menuService ;
    @Mock
    private MenuRepository menuRepository;
    @Mock
    private CategorieRepository categorieRepository;
    @Mock
    private AllergeniciService allergeniciService;
    @Mock
    private ElementiRepository elementiRepository;
    private Elementi elementi;
    private Categorie categoria;
    private List<String> allergenici;

    private Menu menu;

    @BeforeEach
    public void init() {
        menu = new Menu();
        menu.setIdMenu(1L);
        menu.setNomeMenu("Ratatouille23");

        elementi = new Elementi();
        elementi.setNomeElemento("elementi corretta");
        elementi.setDescrizione("descrizione corretta");
        elementi.setPrezzo(BigDecimal.TEN);

        categoria = new Categorie();
        categoria.setNomeCategoria("Categoria giusta");
        categoria.setMenu(menu);
        categoria.setOrdine(1);

        allergenici = new ArrayList<>();
        allergenici.add("Allergenico1");
        allergenici.add("Allergenico2");
    }

    @Test
    public void test_CreaCategoriaConElemento_WhiteBox_Path_1() {

        BadRequestException exception = assertThrows(BadRequestException.class, () ->
        {
            categorieService.creaCategoriaConElemento("", elementi, null);
        });
        assertEquals(HttpStatus.BAD_REQUEST, exception.getStatus());
        assertEquals("Nome categoria non specificato", exception.getMessage());
    }
}

```

```

    }

    @Test
    public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2() {
        when(categorieRepository.existsByNomeCategoriaIgnoreCase(categoria.
            getNomeCategoria())).thenReturn(true);

        BadRequestException exception = assertThrows(BadRequestException.class, () ->
        {
            categorieService.creaCategoriaConElemento(categoria.getNomeCategoria(),
                elementi, null);
        });

        assertEquals("Categoria già presente", exception.getMessage());
    }

    @Test
    public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3() throws
        BadRequestException {
        elementi = null;
        when(categorieService.existsByNomeCategoriaIgnoreCase(anyString())).thenReturn
            (false);
        when(elementiService.controlloCampiElementoValidati(elementi)).thenReturn(true
        );
        BadRequestException exception = assertThrows(BadRequestException.class, () ->
        {
            categorieService.creaCategoriaConElemento(categoria.getNomeCategoria(),
                elementi, null);
        });

        assertEquals("Uno o più campi della elemento non sono stati definiti",
            exception.getMessage());
    }

    @Test
    public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4() {
        when(elementiService.checkIfElementoExists(elementi.getNomeElemento())).
            thenReturn(true);

        BadRequestException exception = assertThrows(BadRequestException.class, () ->
        {
            categorieService.creaCategoriaConElemento(categoria.getNomeCategoria(),
                elementi, null);
        });

        assertEquals("Elemento già esistente", exception.getMessage());
    }

    @Test
    public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5() {
        elementi.setPrezzo(BigDecimal.valueOf(-150));

        BadRequestException exception = assertThrows(BadRequestException.class, () ->
        {

```

```

        categorieService.creaCategoriaConElemento(categoria.getNomeCategoria(),
            elementi, null);
    });

    assertEquals("Il prezzo deve essere maggiore di zero", exception.getMessage())
        ;
    }

}

@Test
public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6() throws
    NotFoundException {
    String nomeCategoria = "Categoria di test";
    String nomeMenu = "Ratatouille23";
    when(menuService.findByNameMenu(nomeMenu)).thenThrow( new NotFoundException( "Menu non trovato"));
    NotFoundException exception = assertThrows(NotFoundException.class, () -> {
        categorieService.creaCategoriaConElemento(nomeCategoria, elementi, null);
    });

    assertEquals("Menu non trovato", exception.getMessage());
}

@Test
public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_9() throws
    BadRequestException, NotFoundException {
    when(menuService.findByNameMenu(menu.getNameMenu())).thenReturn(menu);

    when(categorieRepository.findAll()).thenReturn(new ArrayList<>() {{
        add(categoria);
    }});

    when(categorieRepository.findByOrderByOrdineDesc()).thenReturn(new ArrayList
        <>() {{
        add(categoria);
    }});
    when(categorieRepository.findByNameCategoria(categoria.getNomeCategoria())).thenReturn(Optional.of(categoria));
    ResponseEntity<ApiSuccess> response = categorieService.
        creaCategoriaConElemento(categoria.getNomeCategoria(), elementi, null);

    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType
        ());
    assertEquals("Categoria e Elemento creati con successo", response.getBody().getMessage());
}

@Test
public void test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_9() throws
    BadRequestException, NotFoundException {
    when(menuService.findByNameMenu(menu.getNameMenu())).thenReturn(menu);

    when(categorieRepository.findByNameCategoria(categoria.getNomeCategoria())).thenReturn(Optional.of(categoria));
    ResponseEntity<ApiSuccess> response = categorieService.

```

```

        creaCategoriaConElemento(categoria.getNomeCategoria(), elementi, null);

        assertEquals(HttpStatus.CREATED, response.getStatusCode());
        assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType());
        assertEquals("Categoria e Elemento creati con successo", response.getBody().getMessage());

    }

@Test
public void
    test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_9_10_11_12_14()
throws BadRequestException, NotFoundException {

    when(menuService.findByNameMenu(menu.getNameMenu())).thenReturn(menu);
    when(categorieRepository.findAll()).thenReturn(new ArrayList<>() {{
        add(categoria);
    }});

    when(categorieRepository.findByOrderByOrdineDesc()).thenReturn(new ArrayList<>() {{
        add(categoria);
    }});
    when(categorieRepository.findByNameCategoria(categoria.getNameCategoria())).thenReturn(Optional.of(categoria));

    when(allergeniciService.findOneByNameAllergenicoIgnoreCase(allergenici.get(0)))
        .thenReturn(new Allergenici() {{
            setIdAllergenico(1L);
            setNomeAllergenico(allergenici.get(0));
        }});
    when(allergeniciService.findOneByNameAllergenicoIgnoreCase(allergenici.get(1)))
        .thenReturn(new Allergenici() {{
            setIdAllergenico(2L);
            setNomeAllergenico(allergenici.get(1));
        }});

    ResponseEntity<ApiSuccess> response = categorieService.
        creaCategoriaConElemento(categoria.getNameCategoria(), elementi, Optional.of(allergenici));

    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType());
    assertEquals("Categoria e Elemento creati con successo", response.getBody().getMessage());

}

@Test
public void
    test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_9_10_11_13_14()
throws BadRequestException, NotFoundException {

    when(menuService.findByNameMenu(menu.getNameMenu())).thenReturn(menu);
    when(categorieRepository.findAll()).thenReturn(new ArrayList<>() {{
        add(categoria);
    }});
}

```

```

when(categorieRepository.findByOrderByIdDesc()).thenReturn(new ArrayList
    <>() {{
        add(categoria);
    }});
when(categorieRepository.findByNameCategoria(categoria.getNomeCategoria())).thenReturn(Optional.of(categoria));
ResponseEntity<ApiSuccess> response = categorieService.
    creaCategoriaConElemento(categoria.getNomeCategoria(), elementi, Optional.
    of(allergenici));

assertEquals(HttpStatus.CREATED, response.getStatusCode());
assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType
    ());
assertEquals("Categoria e Elemento creati con successo", response.getBody().
    getMessage());

}

@Test
public void
    test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_10_11_12_14()
throws BadRequestException, NotFoundException {
when(menuService.findByNomeMenu(menu.getNomeMenu())).thenReturn(menu);

when(categorieRepository.findByNameCategoria(categoria.getNomeCategoria())).thenReturn(Optional.of(categoria));
when(allergeniciService.findOneByNomeAllergenicoIgnoreCase(allergenici.get(0))
    ).thenReturn(new Allergenici() {{
        setIdAllergenico(1L);
        setNomeAllergenico(allergenici.get(0));
    }});
when(allergeniciService.findOneByNomeAllergenicoIgnoreCase(allergenici.get(1))
    ).thenReturn(new Allergenici() {{
        setIdAllergenico(2L);
        setNomeAllergenico(allergenici.get(1));
    }});

ResponseEntity<ApiSuccess> response = categorieService.
    creaCategoriaConElemento(categoria.getNomeCategoria(), elementi, Optional.
    of(allergenici));

assertEquals(HttpStatus.CREATED, response.getStatusCode());
assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType
    ());
assertEquals("Categoria e Elemento creati con successo", response.getBody().
    getMessage());

}

@Test
public void
    test_CreaCategoriaConElemento_WhiteBox_Path_1_2_3_4_5_6_7_8_10_11_13_14()
throws NotFoundException, BadRequestException {
when(menuService.findByNomeMenu(menu.getNomeMenu())).thenReturn(menu);
when(categorieRepository.findByNameCategoria(categoria.getNomeCategoria())).thenReturn(Optional.of(categoria));
when(allergeniciService.findOneByNomeAllergenicoIgnoreCase(allergenici.get(0))
    ).thenReturn(null);
when(allergeniciService.findOneByNomeAllergenicoIgnoreCase(allergenici.get(1))
    ).thenReturn(null);

```

```
    ResponseEntity<ApiSuccess> response = categorieService.  
        creaCategoriaConElemento(categoria.getNomeCategoria(), elementi, Optional.  
            of(allergenici));  
    assertEquals(HttpStatus.CREATED, response.getStatusCode());  
    assertEquals(MediaType.APPLICATION_JSON, response.getHeaders().getContentType  
        ());  
    assertEquals("Categoria e Elemento creati con successo", response.getBody()  
        .getMessage());  
  
}  
}
```

5.2.2 Testing WhiteBox reImpostaPassword

Adesso verrà riportato il metodo da testare:

```
public ResponseEntity<Utente> reImpostaPassword(String email, String nuovaPassword)
    throws NotFoundException, InternalServerException {
    Utente utente = findByEmail(email);
    if (utente.getPrimoAccesso() != null) {
        throw new InternalServerException( "Errore: primo accesso già effettuato in precedenza");
    }
    if (!patternPassword(nuovaPassword)) {
        throw new InternalServerException( "Formato password non valido");
    }
    if (passwordEncoder.matches(nuovaPassword,utente.getPassword())) {/
        throw new InternalServerException( "La password deve essere diversa da quella inserita dall'Amministratore");
    }
    utente.setPassword(passwordEncoder.encode(nuovaPassword));
    log.info("data primo Accesso : {}", new Date());
    utente.setPrimoAccesso(new Date());
    return ResponseEntity.ok(saveUtente(utente));
}
```

Alla fine di testare il metodo è stato disegnato il suo grafo del flusso di controllo:

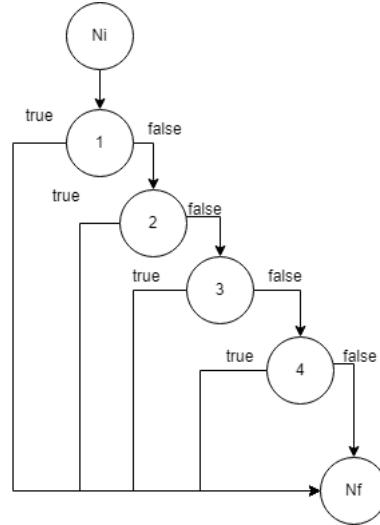


Figura 62: CFG - reImpostaPassword

Dal Control Flow Graph ricaviamo i seguenti metodi al fine di ottenere una **Branch Coverage**:

- test_reImpostaPassword_path_1
- test_reImpostaPassword_path_1_2
- test_reImpostaPassword_path_1_2_3
- test_reImpostaPassword_path_1_2_3_4
- test_reImpostaPassword_path_1_2_3_4_5

```

@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class ReImpostaPasswordWhiteBox {

    @InjectMocks
    UtenteService utenteService;
    @Mock
    UtenteRepository utenteRepository;
    private Utente utente;
    private String email;
    private String password;
    @Mock
    private PasswordEncoder passwordEncoder;
    @BeforeEach
    public void init(){
        utente= new Utente();
        email="ingsw23@gmail.com";
        utente=new Utente();
        utente.setEmail(email);
        utente.setNome("ciro");
        utente.setCognome("de Cristofaro");

        utente.setPassword(passwordEncoder.encode("Password72!"));
        utente.setTipoUtente(TipoUtente.Amministratore);
        utente.setPrimoAccesso(new Date());
        password="Password7!";
    }

    @Test
    public void test_reImpostaPassword_path_1(){
        when(utenteRepository.findByEmail(email)).thenReturn(Optional.empty());

        NotFoundException exception = assertThrows(NotFoundException.class, () -> {
            utenteService.reImpostaPassword(email,password);
        });
        assertEquals("Utente non trovato", exception.getMessage());
    }

    @Test
    public void test_reImpostaPassword_path_1_2(){
        when(utenteRepository.findByEmail(email)).thenReturn(Optional.of(utente));

        InternalServerException exception = assertThrows(InternalServerException.class
            , () -> {
                utenteService.reImpostaPassword(email,password);
            });
        assertEquals("Errore: primo accesso già effettuato in precedenza", exception.
            getMessage());
    }

    @Test
    public void test_reImpostaPassword_path_1_2_3(){
        when(utenteRepository.findByEmail(email)).thenReturn(Optional.of(utente));
        utente.setPrimoAccesso(null);
        password="PasswordNonValida!";
        InternalServerException exception = assertThrows(InternalServerException.class
            , () -> {
                utenteService.reImpostaPassword(email,password);
            });
        assertEquals("Formato password non valido", exception.getMessage());
    }
}

```

```

    @Test
    public void test_reImpostaPassword_path_1_2_3_4(){
        when(utenteRepository.findByEmail(email)).thenReturn(Optional.of(utente));
        utente.setPrimoAccesso(null);
        utente.setPassword(passwordEncoder.encode(password));
        when(passwordEncoder.matches(password,utente.getPassword())).thenReturn(true);
        InternalServerException exception = assertThrows(InternalServerException.class
            , () -> {
                utenteService.reImpostaPassword(email,password);
            });

        assertEquals("La password deve essere diversa da quella inserita dall'Amministratore", exception.getMessage());
    }

    @Test
    public void test_reImpostaPassword_path_1_2_3_4_5() throws InternalServerException
        , NotFoundException {
        when(utenteRepository.findByEmail(email)).thenReturn(Optional.of(utente));
        utente.setPrimoAccesso(null);
        ResponseEntity<Utente> response=utenteService.reImpostaPassword(email,password
            );
        assertEquals(HttpStatus.OK,response.getStatusCode());
    }
}

```