

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
FACOLTÀ DI INFORMATICA



Ingegneria del Software 2022/2023

Ratatouille23

Docenti:

Prof. Sergio Di Martino

Prof. Francesco Cutugno

Prof. Luigi Lucio Libero Starace



Gruppo INGSW2223_N_04

No.	Nome e Cognome	Matricola
1	Luigi Vessella	N86003354
2	Matteo Marino	N86003963
3	Biagio Speranza	N86002964



Indice

1 Descrizione del progetto	5
1.1 Premessa e presentazione	6
2 Documento dei requisiti software	7
2.1 Individuazione target d'utenti	7
2.2 Requisiti Funzionali	7
2.2.1 Admin	7
2.2.2 Cameriere	10
2.2.3 Cucina	10
2.2.4 Supervisore	11
2.2.5 Tutti	11
2.3 Requisiti Non Funzionali	13
2.4 Requisiti di Dominio	13
2.5 Modellazione dei casi d'uso	14
2.5.1 Use-Case Diagram	14
2.5.2 Tabelle di Cockburn	21
2.6 Mock-up dell'applicazione	28
2.6.1 Homepage dell'applicazione	29
2.6.2 Schermata di accesso nel sistema	31
2.6.3 Schermata di registrazione nel sistema	32
2.6.4 Schermata home per gli admin	33
2.6.5 Schermata di registrazione di un nuovo ristorante	34
2.6.6 Schermata di modifica di un ristorante	35
2.6.7 Schermata di registrazione di un nuovo dipendente	36
2.6.8 Schermata di gestione del menù	37
2.6.9 Schermata home per i supervisori	38
2.6.10 Schermata di conferma di uscita	39
2.6.11 Schermata home per i camerieri	40
2.6.12 Schermata di visualizzazione dello stato degli ordini	41
2.6.13 Schermata di visualizzazione del profilo	42
2.6.14 Schermata di cambio password per gli admin	43
2.6.15 Schermata di cambio email per gli admin	44
2.6.16 Schermata di cambio password per i dipendenti al primo accesso	45
2.6.17 Schermata di creazione degli avvisi	46
2.6.18 Schermata di aggiunta piatti al menù	47
2.6.19 Schermata di aggiunta ordini	49
2.6.20 Schermata di visualizzazione delle statistiche	50
2.6.21 Schermata di visualizzazione delle notifiche	51
2.7 Valutazione dell'usabilità (a priori)	52
2.8 Individuazione target d'utenti	54
2.9 Glossario	55
2.10 Class diagram di analisi o dominio	56
2.10.1 Class diagram Login	56



2.10.2 Class diagram Ristorante	57
2.10.3 Class diagram dipendenti	58
2.10.4 Class diagram Menu	59
2.10.5 Class diagram Avvisi	60
2.10.6 Class diagram Ordini	61
2.10.7 Class diagram statistiche	62
2.11 Sequence di analisi	63
2.12 Prototipazione funzionale via statechart dell'interfaccia grafica	65
2.12.1 Aggiungi ristorante	65
2.12.2 Aggiungi piatto	66
2.12.3 Prendi ordine	67
2.12.4 Visualizza avvisi	68
3 Documento di design	69
3.1 Architettura del sistema	69
3.1.1 Client	69
3.1.2 Server	70
3.2 Sequence diagram di design	73
3.2.1 Funzionalità: "Aggiungi Ristorante"	73
3.2.2 Funzionalità: "Aggiungi portata"	73
3.2.3 Funzionalità: "Prendi ordinazione"	74
3.2.4 Funzionalità: "Visualizza avvisi"	74
3.3 Class diagram di design	75
3.3.1 Espansione modelli Lavoratore	75
3.3.2 Login	75
3.3.3 Gestione piatti	76
3.3.4 Gestione Avvisi	76
3.3.5 Gestione Ordini	77
3.3.6 Gestione Dipendenti	77
3.3.7 Visualizzazione Statistiche	78
4 Testing JUnit	79
4.1 Test funzione check credenziali	79
4.2 Test funzione che calcola incasso in un range di date.	82
4.3 Test funzione che controlla i campi di un ristorante.	87
4.4 Testing delle funzione media (in statistiche).	90
4.5 Usabilità sul campo	94
4.5.1 Premessa	94
4.5.2 Presentazione degli utenti	94
4.5.3 Il confronto con gli utenti	94
4.5.4 Valutazione finale	95



1 Descrizione del progetto

Ratatouille23 è un sistema finalizzato al supporto alla gestione e all'operatività di attività di ristorazione. Il sistema consiste in un'applicazione performante e affidabile, attraverso cui gli utenti possono fruire delle funzionalità del sistema in modo intuitivo, rapido e piacevole. La nostra visione della richiesta prevede lo sviluppo di un'applicazione mobile su sistema operativo Android che offrirà agli utilizzatori i seguenti servizi:

- *I proprietari (o amministratori) potranno creare account per i dipendenti*
- *I proprietari saranno in grado di gestire uno o più ristoranti*
- *Si avrà la possibilità di visualizzare e/o modificare il menu*
- *I dipendenti (camerieri) saranno in grado di prendere e inoltrare le ordinazioni in cucina*
- *Gli addetti alla cucina potranno avvisare i camerieri nel momento in cui è pronta un'ordinazione*
- *Tutti potranno visionare lo storico delle ordinazioni con i dettagli*

Ovviamente, il tipo di funzionalità messo a disposizione dall'applicativo sarà cambiato dinamicamente a seconda di chi si logga. Gli amministratori e/o supervisori saranno inoltre dotati di tablet con a bordo l'OS di Google Android per una migliore fruizione della loro dashboard.

I dipendenti quali camerieri, operatori di cucina, capisala, saranno dotati di smartphone aziendali sempre con OS Android correttamente configurati per un'ottimale fruizione dell'applicazione.

Tutti i dispositivi dovranno essere in grado di accedere a internet, preferibilmente per tutta la durata del servizio. Il funzionamento del server è invece garantito da servizi allo stato dell'arte quali Microsoft Azure.



1.1 Premessa e presentazione



Figura 1.1: **Grafico:** Evoluzione dei prodotti high-tech secondo D.Norman

Con questo piccolo preambolo vogliamo richiamare l'attenzione sul diagramma di evoluzione dei prodotti software di D.Norman, il quale mostra il ciclo di vita che solitamente questo tipo di prodotto ha. All'inizio avremo un'app più incentrata sulle tecnologie implementate e/o da implementare per "cercare" di raggiungere ciò che l'utente medio richiede dall'applicativo. Successivamente si otterrà il punto di pareggio in cui si soddisfano i bisogni dell'utente tipico, e ciò potrebbe essere considerato un buon punto di equilibrio del sistema. In fine, sempre ammesso che l'app sia sopravvissuta al mercato durante questo ciclo, ci può essere una fase finale con una condizione di iperfunzionalità, dove si cercherà via via di soddisfare le esigenze di una platea di utenti più ampia. Nel caso della nostra applicazione Ratatouille23 descritta in questo documento, pensiamo di distribuire un prodotto nella sua fase di equilibrio (vedere grafico) che certamente soddisfa le esigenze degli attori utilizzatori descritti nei punti successivi, ma che allo stesso tempo può essere facilmente aggiornata e migliorata qualora dovesse aggiungersi nuovi target d'utenti o cambiare gli esistenti.



2 Documento dei requisiti software

2.1 Individuazione target d'utenti

Durante la progettazione di un nuovo software è fondamentale definire qual'è il target di utenti a cui riferirsi.
Da una prima analisi dei casi d'uso è possibile individuare per la nostra applicazione quattro diverse utenze:

- Admin (o proprietario del ristorante)
- Supervisore
- Addetto alla cucina
- Addetti alla sala (principalmente i camerieri)

In seguito verranno studiate le suddette figure e verrà approfondito lo studio del target d'utenti.

2.2 Requisiti Funzionali

Vengono qui presentati i requisiti funzionali dell'applicativo, ossia quei servizi che l'app deve offrire agli utenti:

2.2.1 Admin

ID	Admin_1
Nome	Registrazione account amministratore
Descrizione	Il sistema permette ad un amministratore non registrato di registrarsi alla piattaforma utilizzando: <i>nome, cognome, email, codice fiscale, P.IVA e password</i>

ID	Admin_2
Nome	Modifica account amministratore
Descrizione	Il sistema permette ad un amministratore loggato di modificare i campi del proprio account.

ID	Admin_3
Nome	Registrazione dei dipendenti
Descrizione	Il sistema permette ad un amministratore di creare utenze per i dipendenti non registrati del ristorante, specificandone <i>nome, cognome, email e ruolo</i> .



ID	Admin _ 4
Nome	Modificare/Eliminare account dipendenti
Descrizione	Il sistema permette ad un amministratore loggato di modificare gli account dei propri dipendenti.

ID	Admin _ 5
Nome	Aggiungere personale della cucina
Descrizione	Il sistema permette ad un amministratore loggato di aggiungere al proprio ristorante il personale della cucina.

ID	Admin _ 6
Nome	Aggiunta dei Ristoranti
Descrizione	Il sistema permette ad un amministratore di poter aggiungere le proprie attività di ristorazione (CAMPI DA DEFINIRE).

ID	Admin _ 7
Nome	Modifica/Eliminazione dei Ristoranti
Descrizione	Il sistema permette ad un amministratore di poter modificare ed eliminare le proprie attività di ristorazione del sistema.

ID	Admin _ 8
Nome	Modifica dati dei Dipendenti
Descrizione	Il sistema permette ad un amministratore loggato di poter cambiare i dati personali dei dipendenti (nome, cognome, email, luogo).



ID	Admin_9
Nome	Aggiungere/Modificare elementi nel menù
Descrizione	Il sistema permette ad un amministratore o supervisore dell'attività di ristorazione di aggiungere/modificare elementi nel menù dell'attività. Ogni elemento dovrà avere i seguenti campi: Nome, Costo, Descrizione, Elenco di Allergeni, Categoria/e.

ID	Admin_10
Nome	Modifica dati personali
Descrizione	Il sistema permette ad un amministratore loggato di poter cambiare i propri dati personali.

ID	Admin_12
Nome	Tradurre il menu
Descrizione	Il sistema permette ad un Amministratore di poter tradurre gli elementi (nome e descrizione) del proprio menù in un'altra lingua.

ID	Admin_13
Nome	Visualizza statistiche personale della cucina
Descrizione	Il sistema permette ad un Amministratore di visualizzare, grazie anche all'ausilio di grafici interattivi, informazioni sull'operato degli addetti alla cucina.

ID	Admin_14
Nome	Modifica menu
Descrizione	Il sistema permette ad un Amministratore di poter aggiornare (aggiungere, modificare ed eliminare elementi) il menu del ristorante.



2.2.2 Cameriere

ID	Waiter_1
Nome	Prendere le ordinazioni
Descrizione	Il sistema permette ai camerieri di prendere ordinazioni ai tavoli, inoltrandole alla cucina.

ID	Waiter_2
Nome	Gestione delle ordinazioni
Descrizione	Il sistema permette ai camerieri di prendere ordinazioni ai tavoli, inoltrandole alla cucina.

ID	Waiter_3
Nome	Evasione degli ordini
Descrizione	Il sistema permette a un Cameriere di marcire i singoli elementi di un ordine come conclusi, aggiornando gli addetti in cucina.

ID	Waiter_4
Nome	Sollecitare la cucina
Descrizione	Il sistema permette a un Cameriere di sollecitare la cucina nel caso in cui un ordine sia da troppo tempo in preparazione.

2.2.3 Cucina

ID	Kitchen_1
Nome	Marcare gli ordini pronti
Descrizione	Il sistema permette alla cucina di marcire gli ordini pronti alla "consegna", specificando il nome dello chef che l'ha preparato.

ID	Kitchen_2
Nome	Sollecitare i camerieri
Descrizione	Il sistema permette alla cucina di notificare i camerieri nel caso in cui un ordine sia pronto alla consegna da troppo tempo.



2.2.4 Supervisore

ID	Hypervisor_1
Nome	Avvisare il personale
Descrizione	Il sistema permette ad un supervisore ed un amministratore di inviare degli avvisi al personale.

ID	Hypervisor_2
Nome	Visualizzare stato ordini per tavolo
Descrizione	Il sistema permette ad un Supervisore ed un Cameriere di visualizzare gli stati delle ordinazioni per ogni tavolo.

ID	Hypervisor_3
Nome	Visualizzare ordini in arrivo
Descrizione	Il sistema permette ad un Supervisore e alla Cucina di visualizzare l'elenco di ordini in arrivo.

ID	Hypervisor_4
Nome	Visualizzare ordini in uscita
Descrizione	Il sistema permette ad un Supervisore e alla Cucina di visualizzare l'elenco di ordini pronti all'uscita.

ID	Hypervisor_5
Nome	Visualizzare storico ordini
Descrizione	Il sistema permette ad un Supervisore e alla Cucina di visualizzare lo storico degli ordini.

2.2.5 Tutti

ID	All_1
Nome	Recupero/Cambio Password
Descrizione	Il sistema permette a tutti gli utenti registrati di poter recuperare la password e di poterla modificare.



ID	All_2
Nome	Login
Descrizione	Il sistema permette a tutti gli utenti registrati di poter effettuare il login con Username e Password all'interno della piattaforma.

ID	All_3
Nome	Visualizzare un avviso o sollecitazione
Descrizione	Il sistema permette a tutti gli utenti registrati di poter visualizzare gli avvisi e le sollecitazioni ricevuti, marcandoli come visualizzati.



2.3 Requisiti Non Funzionali

Vengono qui elencati i requisiti non funzionali dell'applicativo:

ID	Unfunctional_1
Nome	Policy first password
Descrizione	Il sistema richiede al primo accesso di un dipendente il cambio della password provvisoria in una password personale.

ID	Unfunctional_2
Nome	Verifica esistenza della mail
Descrizione	Al fine di evitare registrazioni con e-mail fittizie, il sistema richiede l'autenticazione della mail mediante codice di verifica per poter procedere con il completamento della registrazione

ID	Unfunctional_3
Nome	Password Strength
Descrizione	Al fine di evitare la creazione di password poco sicure, il sistema impone all'utente di utilizzare una password di almeno 8 caratteri che contenga numeri e caratteri speciali.

ID	Unfunctional_4
Nome	Unique Commercial Code
Descrizione	Una P.IVA appartiene ad un solo amministratore.

2.4 Requisiti di Dominio

ID	Domain_1
Nome	GDPR
Descrizione	Il sistema deve essere conforme alla normativa GDPR (Regolamento Generale sulla Protezione dei Dati), per il trattamento dei dati personali e riguardante la privacy dell'utente.



2.5 Modellazione dei casi d'uso

In questa sezione vengono riportati i Use-case diagram relativi al sistema, le tabelle di Cockburn e i Sequence diagram relativi alle funzionalità *Aggiungi ristorante*, *Aggiungi piatto*, *Prendi ordinazione*, *Visualizza avvisi*.

2.5.1 Use-Case Diagram

Qui viene riportato lo Use Case Diagram relativo all'applicazione, che per rendere la lettura più semplificata è stato diviso in più sezioni.

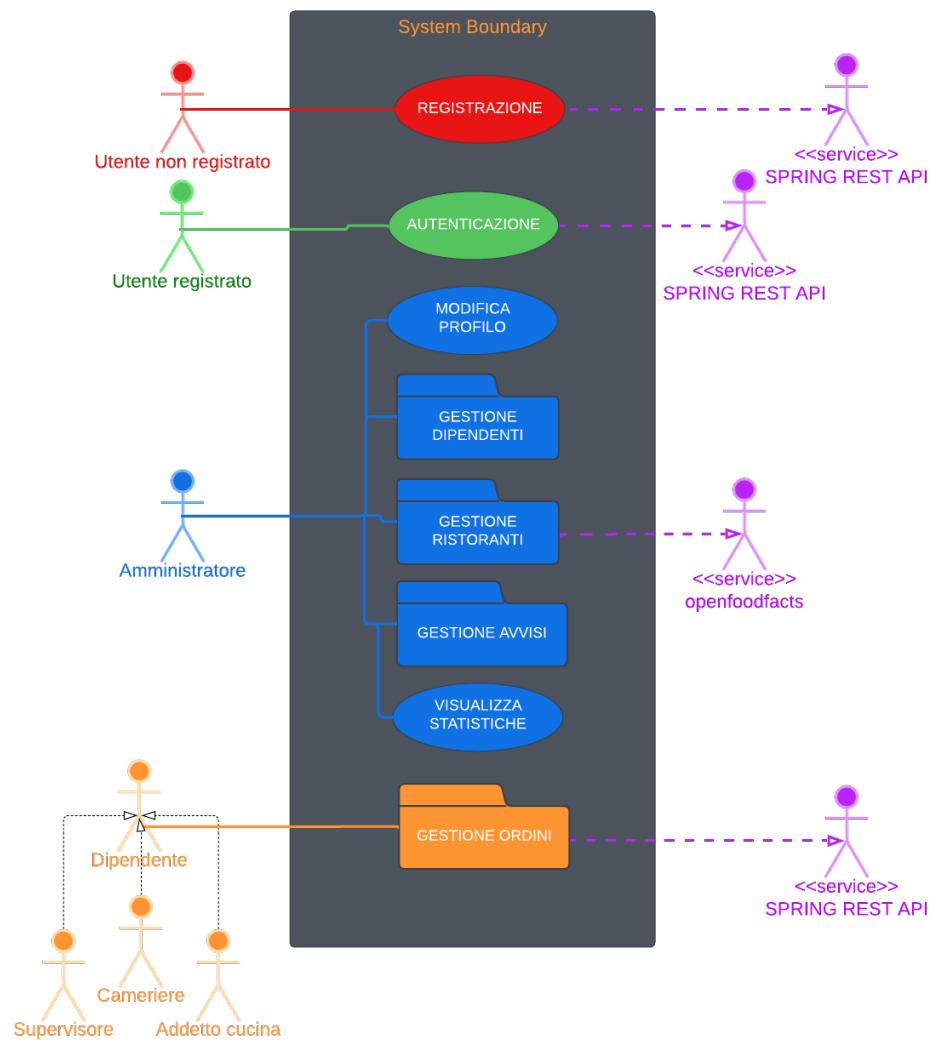


Figura 2.1: Use Case dell'applicazione

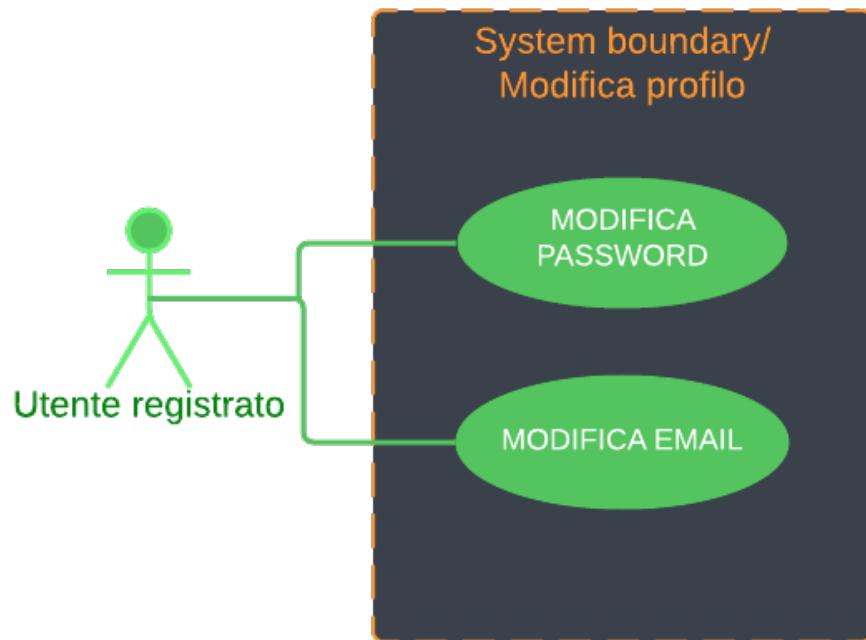


Figura 2.2: Use Case relativo alla modifica del profilo

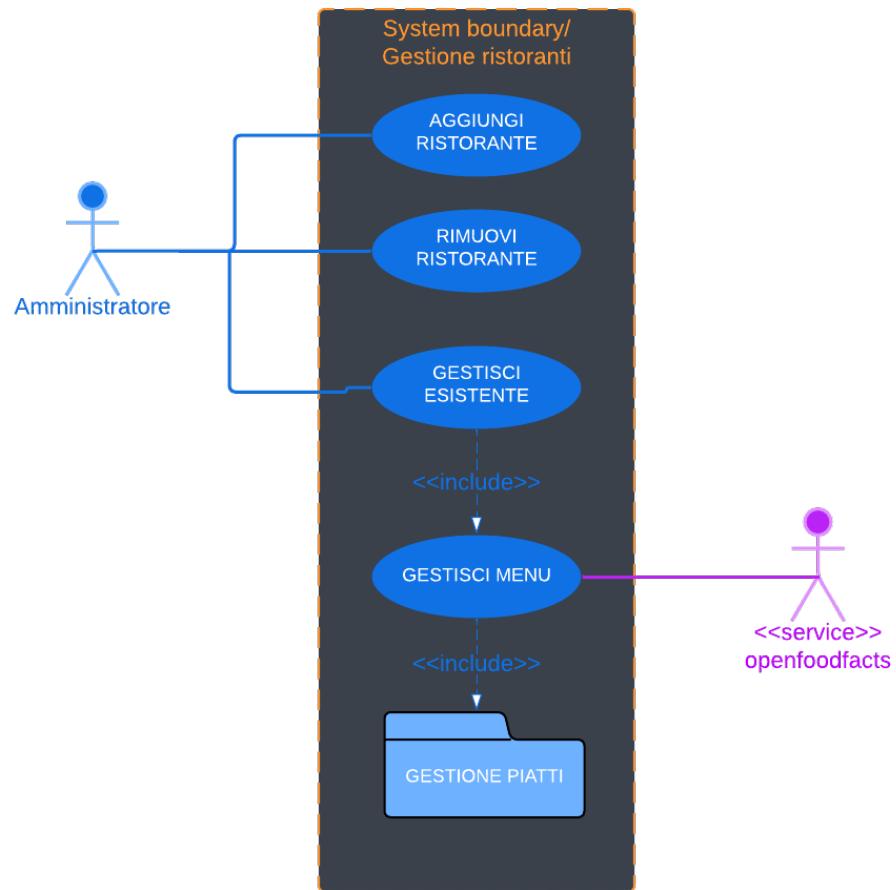


Figura 2.3: Use Case relativo alla gestione dei ristoranti

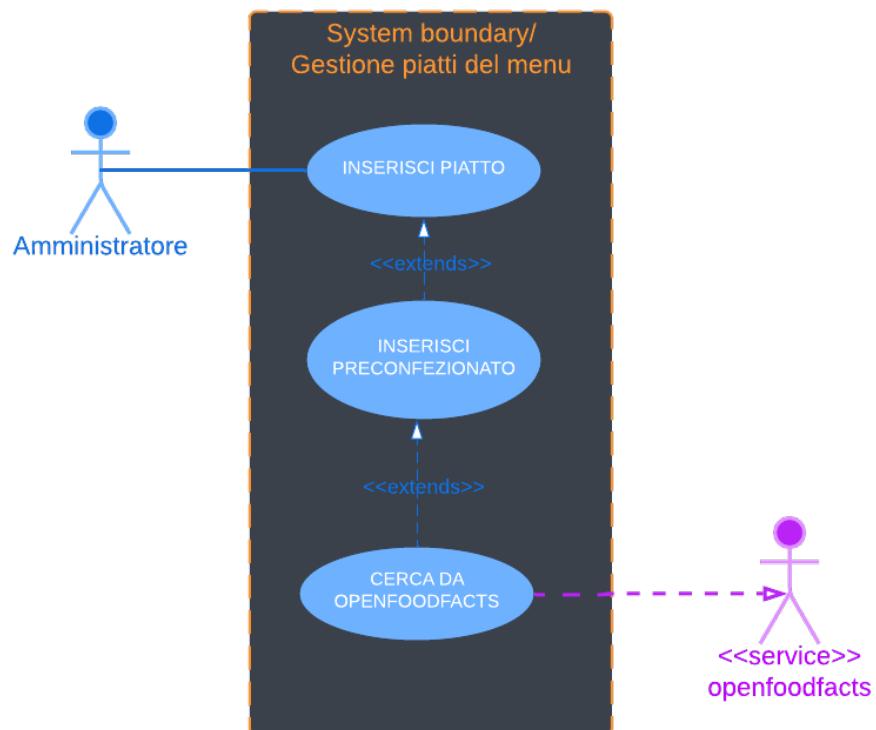


Figura 2.4: Use Case relativo alla gestione dei piatti del menù

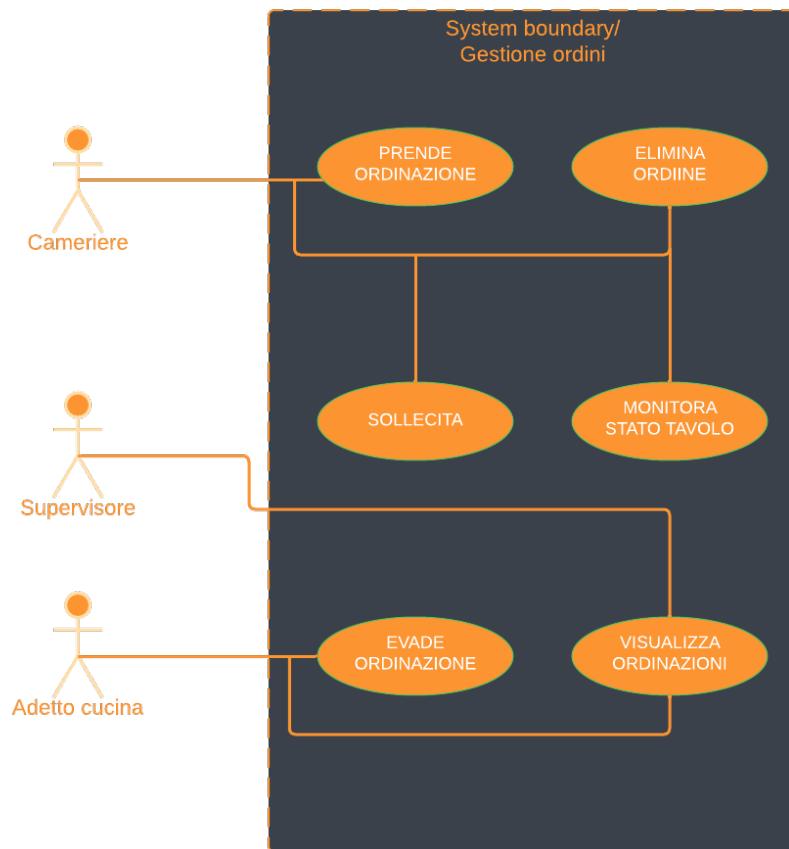


Figura 2.5: Use Case relativo alla gestione degli ordini

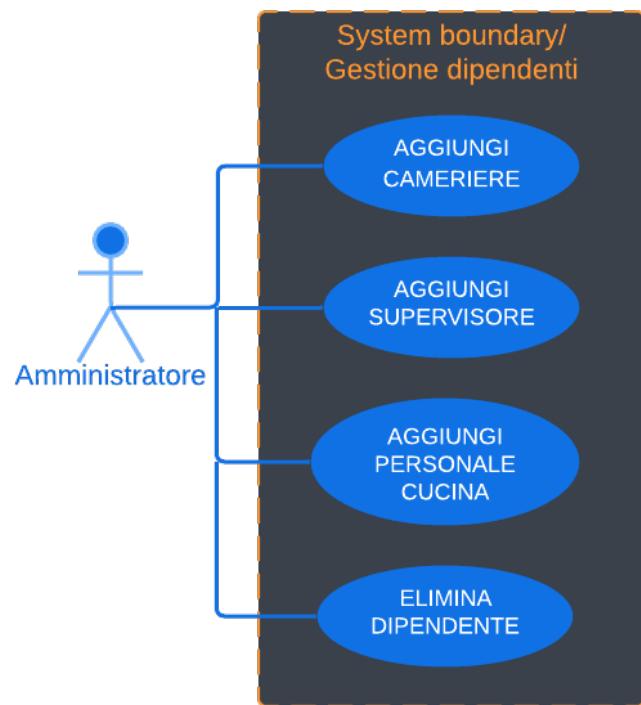


Figura 2.6: Use Case relativo alla gestione dei dipendenti

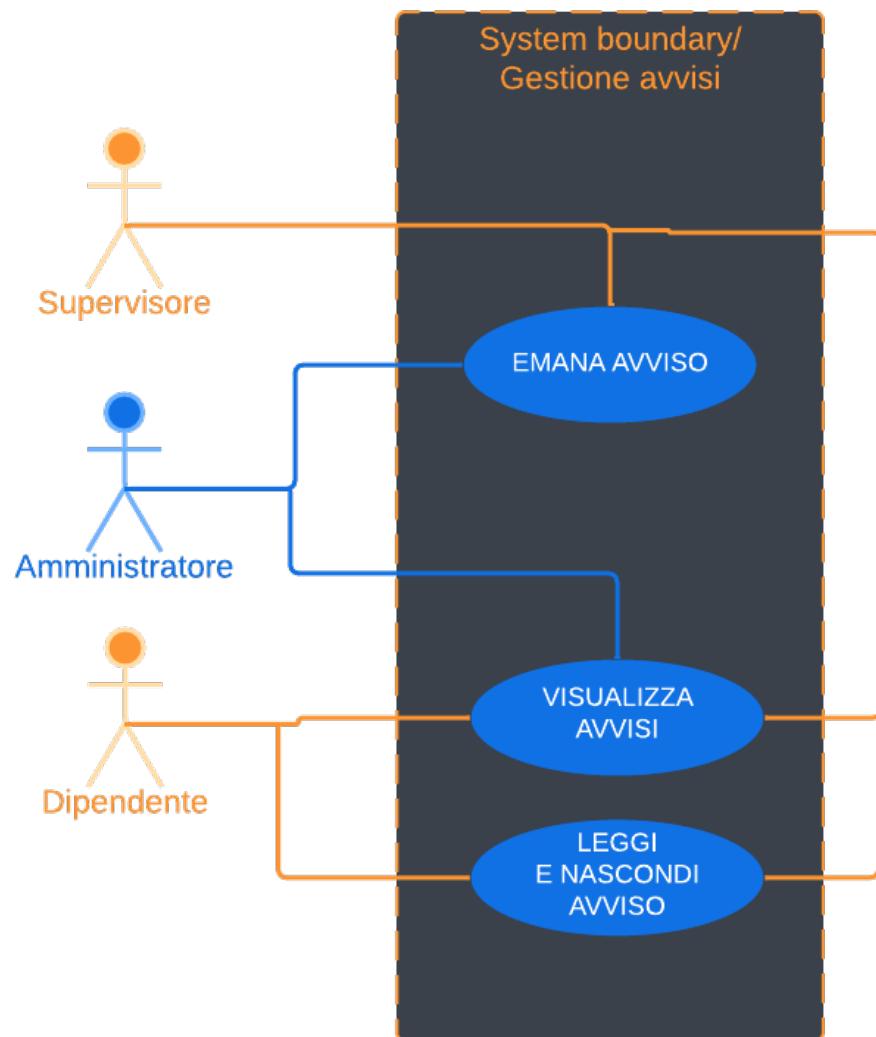


Figura 2.7: Use Case relativo alla gestione degli avvisi



2.5.2 Tabelle di Cockburn

Vengono qui riportate le tabelle di cockburn dei casi d'uso *Aggiungi ristorante*, *Aggiungi piatto*, *Prendi ordinazione*, *Visualizza avvisi*.

Tabella 1:

Use Case #1	Aggiunta ristorante		
Goal in Context	Un admin (proprietario di uno o più ristoranti) vuole aggiungere uno di questi nell'app Ratatouille.		
Precodition	Il proprietario deve essere registrato e loggato nell'app come amministratore		
Success End Condition	Il proprietario aggiunge correttamente il ristorante		
Failed End Condition	Il proprietario non riesce ad aggiungere il ristorante		
Primary Actor	Amministratore		
Trigger	Preme il pulsante <i>Aggiungi</i>		
Description	Step	UserAction	System
	1	L'amministratore preme sul tasto "AGGIUNGI RISTORANTE" sulla schermata M04	
	2		Mostra la schermata M05
	3	Compila i campi necessari per la registrazione del proprio ristorante e preme sul tasto "SALVA" per aggiungere il Ristorante	
	4		Ricarica la schermata M04 aggiungendo nella lista dei ristoranti l'ultimo appena inserito
Extension # 1 L'amministratore non fa nulla e torna indietro	Step	UserAction	System
	4a		Torna alla schermata M04

Continued on next page



Tabella 1: (Continued)

Use Case #1	Aggiunta ristorante		
Extension #2 L'amministratore ha aggiunto un ristorante già presente nella propria lista di ristoranti	Step	UserAction	System
	4b		Mostra uno dei messaggi di errore della schermata M05 restando allo step 2 dello scenario principale
Extension #3 L'amministratore ha lasciato uno o più campi vuoti	Step	UserAction	System
	4c		Mostra uno o più dei messaggi di errore della schermata M05 restando allo step 2 dello scenario principale
Extension #4 L'amministratore ha aggiunto un nome troppo corto	Step	UserAction	System
	4d		Mostra il messaggio di errore di fianco al campo "Nome" della schermata M05 restando allo step 2 dello scenario principale
Extension #5 L'amministratore ha aggiunto un numero di coperti non valido	Step	UserAction	System
	4e		Mostra il messaggio di errore di fianco al campo "Numero di coperti" della schermata M05 restando allo step 2 dello scenario principale
Extension #6 L'amministratore ha aggiunto un indirizzo troppo corto	Step	UserAction	System
	4f		Mostra il messaggio di errore di fianco al campo "Indirizzo" della schermata M05 restando allo step 2 dello scenario principale

Continued on next page



Tabella 1: (Continued)

Use Case #1	Aggiunta ristorante		
	Step	UserAction	System
Extension #7 L'amministratore ha aggiunto un numero di telefono non valido	4g		Mostra il messaggio di errore di fianco al campo "Numero di telefono" della schermata M05 restando allo step 2 dello scenario principale
Subvariation #1 L'amministratore torna indietro completando solo parzialmente i campi	Step	UserAction	System
	1		Mostra la schermata M10
	2	Preme su "SI"	
	3		Torna alla schermata M04
Subvariation #2 L'amministratore inizialmente vuole tornare indietro completando solo parzialmente i campi ma cambia idea	Step	UserAction	System
	1		Mostra la schermata M10
	2	Preme su "NO"	
	3		Torna alla schermata M05 allo step 2 dello scenario principale



Tabella 2:

Use Case #2	Prendere un ordine		
Goal in Context	Un cameriere vuole prendere l'ordinazione di un tavolo		
Precodition	Il cameriere deve essere registrato e loggato nell'app e il ristorante deve avere un menu con dei piatti al suo interno		
Success End Condition	Il cameriere prendere correttamente un'ordinazione		
Failed End Condition	Il cameriere non prende l'ordinazione		
Primary Actor	Cameriere		
Trigger	Preme il pulsante <i>NUOVO ORDINE</i>		
Description	Step	UserAction	System
	1	Il cameriere preme sul pulsante " <i>NUOVO ORDINE</i> " sulla schermata M11	
	2		Mostra la schermata M19 con i vari piatti del menu
	3	Il cameriere seleziona il numero del tavolo, inserisce le portate all'interno dell'ordine e preme il pulsante " <i>SALVA</i> "	
	4		Torna alla schermata M11
Extension # 1 Il cameriere non fa nulla e torna indietro	Step	UserAction	System
	4a		Torna alla schermata M11
Subvariation #1 Il cameriere torna indietro dopo aver aggiunto dei piatti all'ordine senza salvare	Step	UserAction	System
	1		Mostra la schermata M10
	2	Preme su "SI"	
	3		Torna alla schermata M11

Continued on next page



Tabella 2: (Continued)

Use Case #2	Prendere un ordine		
Subvariation #2	Step	UserAction	System
Il cameriere inizialmente vuole tornare indietro, con dei piatti aggiunti all'ordinazione, ma cambia idea	1		Mostra la schermata M10
	2	Preme su "NO"	
	3		Torna alla schermata M11

Tabella 3:

Use Case #3	Aggiungi piatto al menu		
Goal in Context	Un amministratore vuole aggiungere un piatto al proprio menu		
Precodition	Il proprietario deve essere registrato e loggato nell'app come amministratore, deve aver aggiunto almeno un ristorante e deve averne creato un menu		
Success End Condition	L'amministratore aggiunge correttamente un piatto al suo menu		
Failed End Condition	L'amministratore non riesce ad aggiungere un piatto al suo ristorante		
Primary Actor	Amministratore		
Trigger	Preme il pulsante <i>Aggiungi prodotto</i>		
Description	Step	UserAction	System
L'amministratore non fa nulla e torna indietro	1	L'amministratore preme sul pulsante <i>Aggiungi prodotto</i> sulla schermata M08	
	2		Mostra la schermata M18
	3	L'amministratore compila i campi del piatto e preme sul pulsante pulsante <i>OK</i>	
	4		Torna alla schermata M08
Extension # 1	Step	UserAction	System
L'amministratore non fa nulla e torna indietro	4a		Torna alla schermata M08

Continued on next page



Tabella 3: (Continued)

Use Case #3	Aggiungi piatto al menu		
Extension # 2	Step	UserAction	System
L'amministratore compila solo parzialmente i campi e preme sul pulsante <i>OK</i>	4a		Mostra l'errore nella schermata M18
Subvariation #1 L'amministratore vuole aggiungere un prodotto preconfezionato al proprio menu	Step	UserAction	System
	1		Mostra la schermata M18
	2	L'amministratore scrive il nome (completo o parziale) di un prodotto e preme il pulsante <i>Cerca</i>	
	3		Mostra i prodotti corrispondenti alla ricerca
	4	L'amministratore seleziona il prodotto desiderato e preme il pulsante <i>OK</i>	
	5		Torna alla schermata M08



Tabella 4:

Use Case #4	Visualizza avvisi		
Goal in Context	Un dipendente vuole visualizzare un avviso		
Precodition	Il dipendente deve essere registrato e loggato nell'app		
Success End Condition	Il dipendente visualizza gli avvisi		
Failed End Condition	Il dipendente non riesce a visualizzare i propri avvisi		
Primary Actor	Dipendente		
Trigger	Preme il pulsante <i>Avvisi</i> nella propria Dashboard		
Description	Step	UserAction	System
	1	Il Dipendente preme sul pulsante <i>Avvisi</i> sulla schermata M11 o M09 o M12	
	2		Mostra la schermata M21
Extension # 1 Non ci sono avvisi da mostrare	Step	UserAction	System
	4a		Mostra la schermata M21
Subvariation #1 Il Dipendente vuole marcare un avviso come visualizzato	Step	UserAction	System
	1	Il Dipendente scorre col dito sull'avviso	
	2		Resta sulla schermata M21 eliminando l'avviso



2.6 Mock-up dell'applicazione

L'interfaccia utente ha il compito di "filtrare" la complessità, presentando all'utente un'immagine semplificata del prodotto, e congruente con i compiti che egli deve svolgere. Una buona interfaccia non solo nasconde la complessità interna del sistema, ma ne riduce la complessità funzionale, mettendo a disposizione dell'utente funzioni di più alto livello, in grado di effettuare compiti complessi con un grado di automatismo maggiore. (da "Facile da Usare")

Di seguito verranno presentati i mockup relativi all'applicazione. (Si noti che in questa fase i prototipi dell'interfaccia grafica potrebbero differire da quella che sarà poi la versione finale dell'applicazione.) Anche in questo caso abbiamo usato tool molto potenti per la prototipazione delle viste dell'app. Più in particolare, ci siamo avvalsi del tool di design **Figma**.

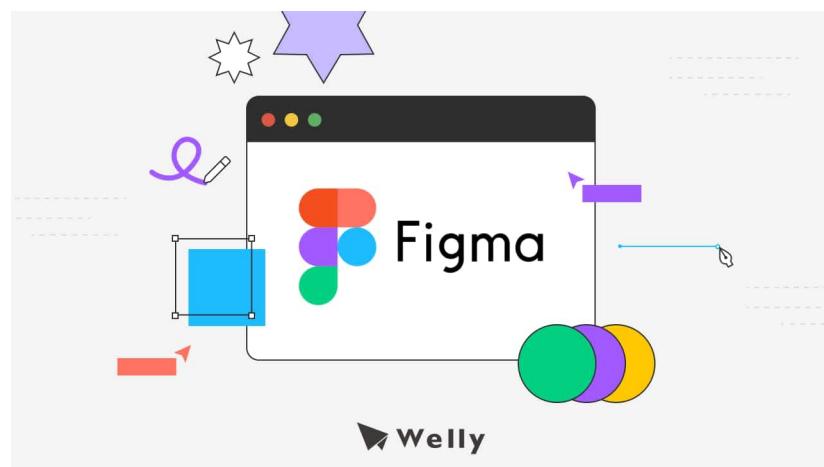


Figura 2.8: Tool Figma

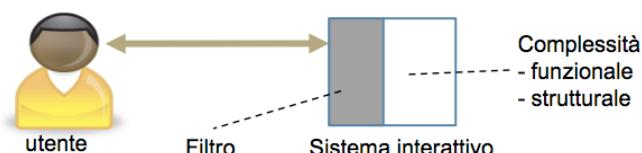


Figura 2.9: L'interfaccia Utente



2.6.1 Homepage dell'applicazione

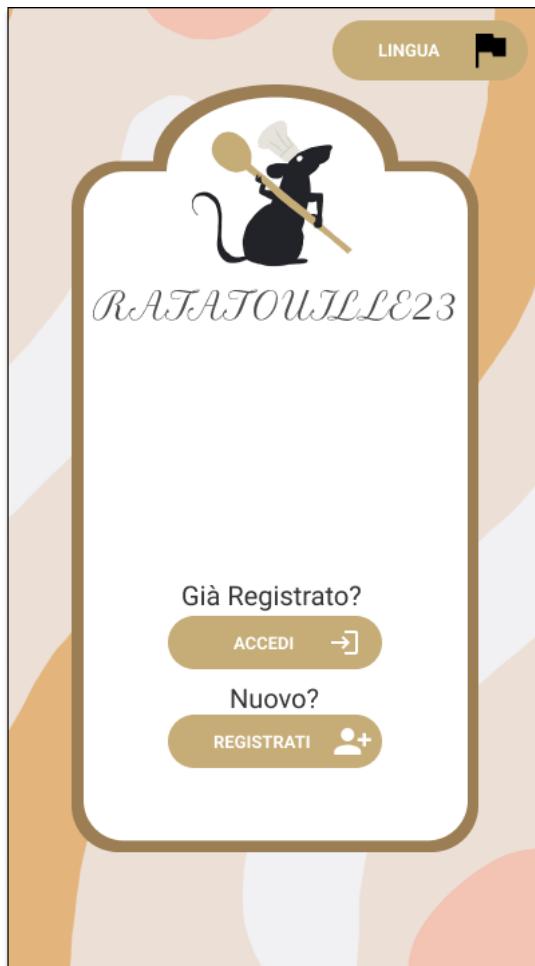


Figura 2.10: M01: Homepage dell'applicazione



ID *M01*

Componenti:

Tipo	Nome	Funzione
Bottone	ACCEDI	Quando cliccato porta alla schermata <i>M02</i>
Bottone	REGISTRATI	Quando cliccato porta alla schermata <i>M03</i>



2.6.2 Schermata di accesso nel sistema

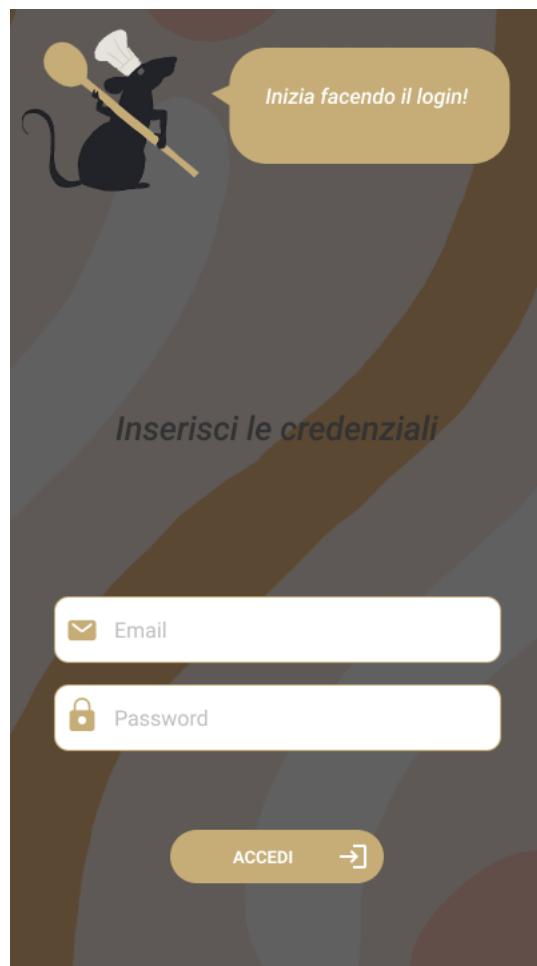


Figura 2.11: **M02**: Schermata di accesso nel sistema

ID **M02**

Componenti:

Tipologia	Nome	Funzione
Edit Text	EMAIL	Permette l'inserimento dell'email dell'utente
Edit Text	PASSWORD	Permette l'inserimento della password dell'utente
Bottone	ENTRA	Quando cliccato porta alla schermata M04 se admin, M09 se supervisore, M11 se cameriere, M12 se account della cucina



2.6.3 Schermata di registrazione nel sistema

Figura 2.12: M03: Schermata di registrazione nel sistema

ID M03

Nota: la schermata di registrazione è valida solo per gli admin proprietari dei ristoranti, in quanto sono poi loro a registrare i dipendenti.

Componenti:

Tabella 5:

Tipologia	Nome	Funzione
Edit Text	NOME	Permette di inserire il nome dell'admin
Edit Text	COGNOME	Permette di inserire il cognome dell'admin
Edit Text	PASSWORD	Permette di inserire una password per l'admin
Edit Text	EMAIL	Permette di inserire l'email dell'admin
Edit Text	CODICE FISCALE	Permette di inserire il codice fiscale dell'admin
Edit Text	P.IVA	Permette di inserire la partita IVA dell'admin
Bottone	REGISTRATI	Quando cliccato riporta alla schermata M01 per permettere l'accesso



2.6.4 Schermata home per gli admin



Figura 2.13: M04: Schermata home per gli admin

ID **M04**

Componenti:

Tabella 6:

Tipologia	Nome	Funzione
ScrollView	I TUOI RISTORANTI	Visualizza ed eventualmente permette la modifica dei ristoranti registrati
Bottone	AGGIUNGI RISTORANTE	Quando cliccato porta alla schermata M05
Bottone	MODIFICA	Quando cliccato porta alla schermata M06
Bottone	PROFILO	Quando cliccato porta alla schermata M13
Bottone	ESCI	Quando cliccato mostra la schermata M10
Bottone	STATISTICHE	Quando cliccato porta alla schermata M20



2.6.5 Schermata di registrazione di un nuovo ristorante



Figura 2.14: M05: Schermata di registrazione di un nuovo ristorante

ID **M05**

Componenti:

Tipo	Nome	Funzione
Edit Text	NOME	Permette di inserire il nome del nuovo ristorante
Edit Text	LOCALITA'	Permette di inserire la località del nuovo ristorante
Edit Text	COPERTI	Permette di inserire il n° dei coperti del nuovo ristorante
Edit Text	NUMERO DI TELEFONO	Permette di inserire il contatto telefonico del ristorante
Bottone	SALVA	Quando cliccato salva il nuovo ristorante nel database e torna alla schermata M04



2.6.6 Schermata di modifica di un ristorante



Figura 2.15: M06: Schermata di modifica di un ristorante

ID **M06**

Componenti:

Tipologia	Nome	Funzione
Bottone	AGGIUNGI DIPENDENTE	Quando cliccato porta alla schermata M07
Bottone	GESTISCI MENU	Quando cliccato porta alla schermata M08
ScrollView	IL TUO PERSONALE	Mostra il personale che lavora nel ristorante, dove se si tiene premuto il bottone <i>Elimina</i> viene eliminato l'account del dipendente scelto dal ristorante
Bottone	CREA AVVISO	Quando cliccato mostra la schermata M17



2.6.7 Schermata di registrazione di un nuovo dipendente

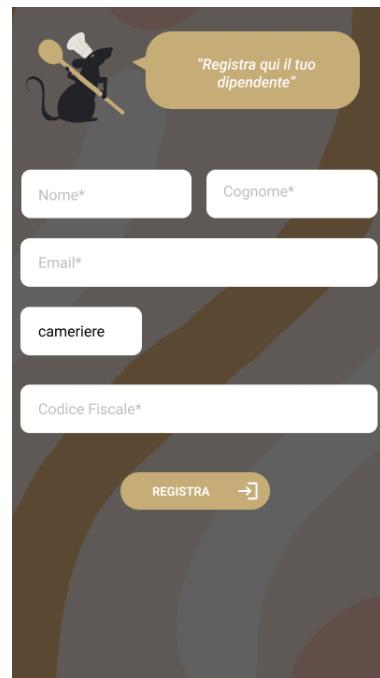


Figura 2.16: M07: Schermata di registrazione di un nuovo dipendente

ID **M07**

Componenti:

Tipo	Nome	Funzione
Edit Text	NOME	Permette di inserire il nome del nuovo dipendente
Edit Text	COGNOME	Permette di inserire il cognome del nuovo dipendente
Edit Text	EMAIL	Permette di inserire l'email del nuovo dipendente
Edit Text	CODICE FISCALE	Permette di inserire il codice fiscale del nuovo dipendente
Spinner	cameriere (default)	Permette di inserire il ruolo del nuovo dipendente
Bottone	REGISTRA	Quando cliccato, se tutti i dati sono corretti, riporta alla schermata M04 registrando il nuovo dipendente



2.6.8 Schermata di gestione del menù

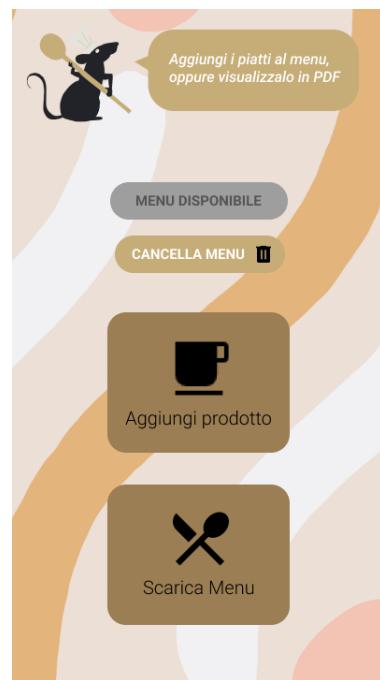


Figura 2.17: M08: Schermata di gestione del menù

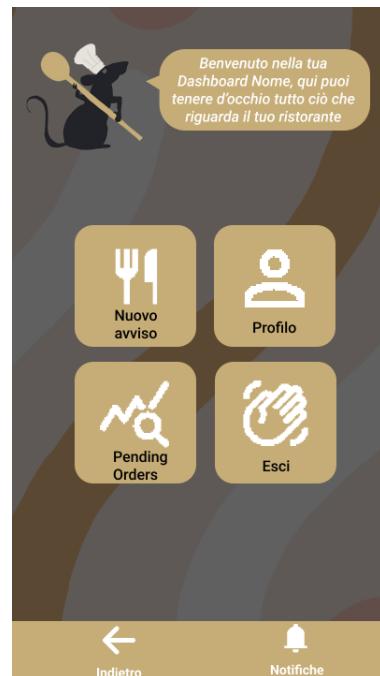
ID **M08**

Componenti:

Tipologia	Nome	Funzione
Bottone	AGGIUNGI PRODOTTO	Quando cliccato porta alla schermata M18
Edit Text	SCARICA MENU	Permette di generare il menù e salvarlo sul dispositivo in formato PDF
Bottone	CREA MENU	Quando cliccato permette di creare un menu (se non è già stato creato)
Bottone	CANCELLA MENU	Permette di cancellare il menu del ristorante (se esistente)



2.6.9 Schermata home per i supervisori



M09: Schermata home dei supervisori

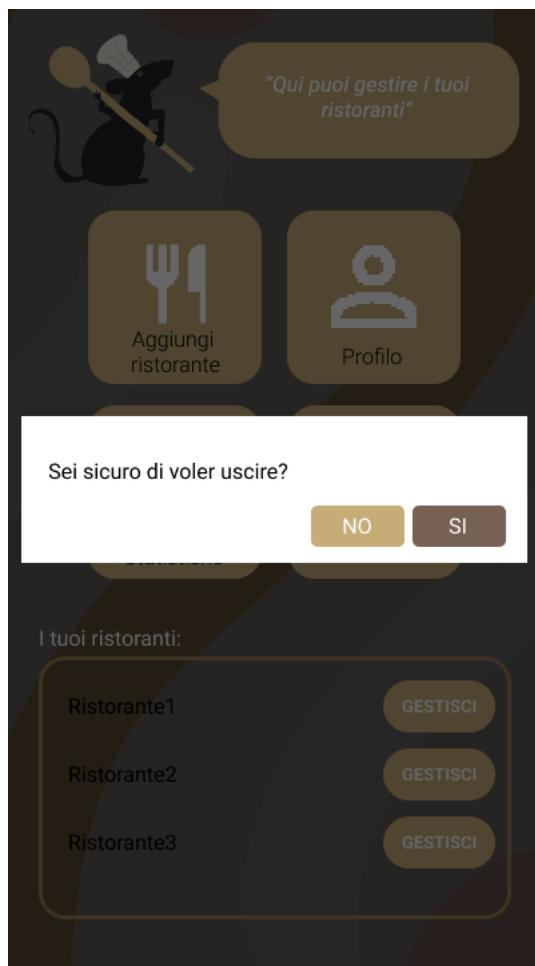
ID **M09**

Componenti:

Tipo	Nome	Funzione
Bottone	PROFILO	Quando cliccato porta alla schermata M13
Bottone	NUOVO AVVISO	Quando cliccato porta alla schermata M21
Bottone	PENDING ORDERS	Quando cliccato porta alla schermata M12
Bottone	ESCI	Quando cliccato porta alla schermata M10
Bottom Navigation Bar	BARRA NAVIGAZIONE	Permette di visualizzare le notifiche cliccando su <i>Notifiche</i> che porterà alla schermata M21 e di tornare indietro cliccando su <i>Indietro</i>



2.6.10 Schermata di conferma di uscita



M10: Schermata di conferma logout

ID **M10**

Componenti:

Tipologia	Nome	Funzione
Bottone	SI	Quando cliccato porta alla schermata M02
Bottone	NO	Quando cliccato resta nella schermata corrente



2.6.11 Schermata home per i camerieri



M11: Schermata home per i camerieri

ID **M11**

Componenti:

Tipologia	Nome	Funzione
Bottone	NUOVO ORDINE	Quando cliccato porta alla schermata M19
Bottone	STATO ORDINE	Quando cliccato porta alla schermata M12
Bottom Navigation Bar	BARRA NAVIGAZIONE	Permette di visualizzare le notifiche cliccando su <i>Notifiche</i> che porterà alla schermata M21 e di tornare indietro cliccando su <i>Indietro</i>



2.6.12 Schermata di visualizzazione dello stato degli ordini



M12: Schermata home per i camerieri

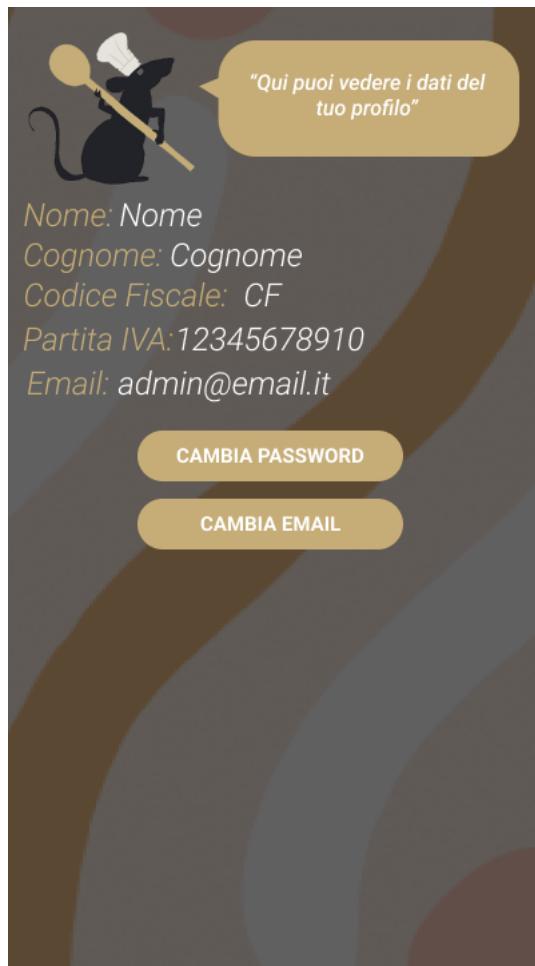
ID **M12**

Componenti:

Tipologia	Nome	Funzione
RecyclerView	ORDINI	Permette di visualizzare la lista degli ordini, dove per i camerieri vi è la possibilità di sollecitare la cucina su un ordine e per la cucina di evaderlo eliminandolo
Bottom Navigation Bar	BARRA NAVIGAZIONE	Permette di visualizzare le notifiche cliccando su <i>Notifiche</i> che porterà alla schermata M21 e di tornare indietro cliccando su <i>Indietro</i>



2.6.13 Schermata di visualizzazione del profilo



M13: Schermata visualizzazione profilo

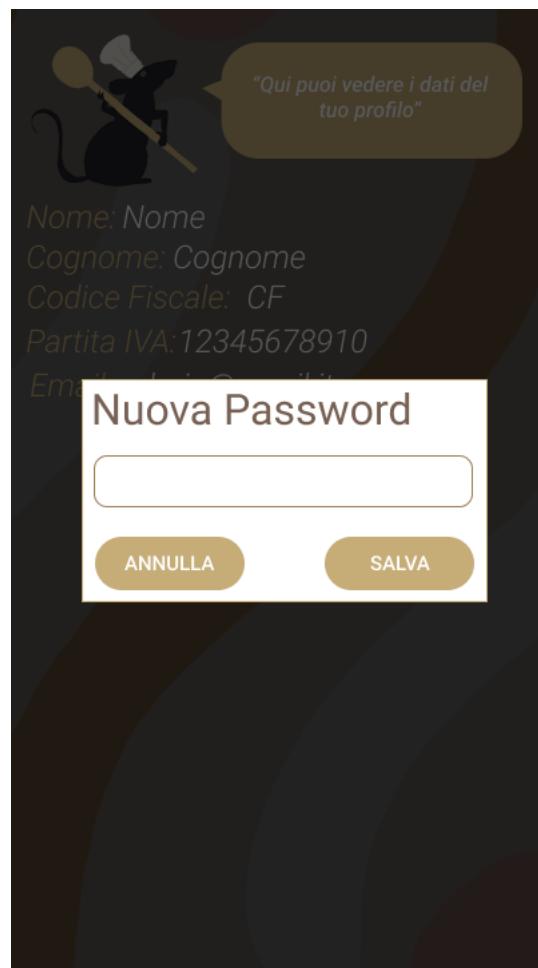
ID **M13**

Componenti:

Tipologia	Nome	Funzione
Bottone	CAMBIA PASSWORD	Quando cliccato mostra la schermata M14
Bottone	CAMBIA EMAIL	Quando cliccato mostra la schermata M15



2.6.14 Schermata di cambio password per gli admin



M14: Schermata cambio password admin

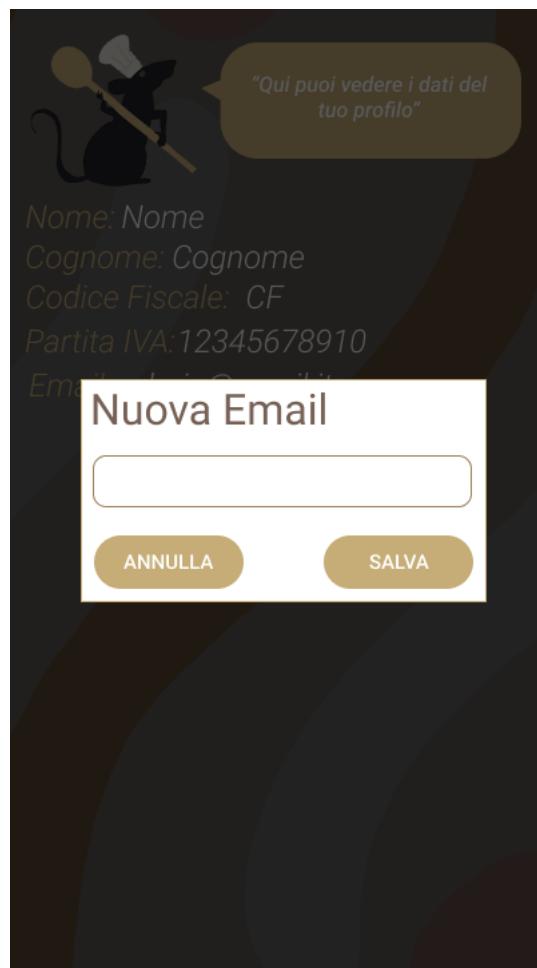
ID **M14**

Componenti:

Tipologia	Nome	Funzione
Edit Text	NUOVA PASSWORD	Permette di inserire la nuova password per l'account dell'admin
Bottone	ANNULLA	Quando cliccato torna alla schermata M13 senza effettuare modifiche
Bottone	SALVA	Quando cliccato torna alla schermata M13 modificando la password dell'account



2.6.15 Schermata di cambio email per gli admin



M15: Schermata cambio email admin

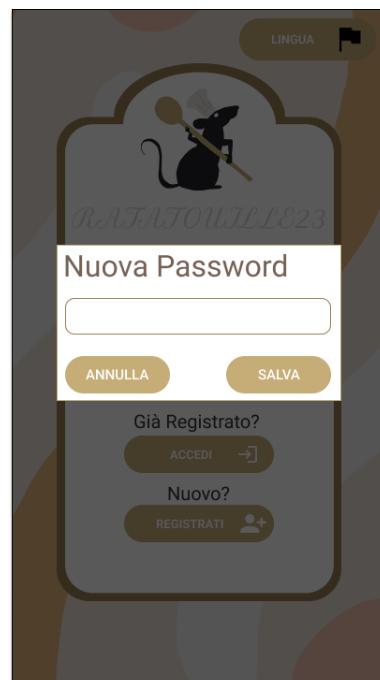
ID **M15**

Componenti:

Tipologia	Nome	Funzione
Edit Text	NUOVA EMAIL	Permette di inserire la nuova email per l'account dell'admin
Bottone	ANNULLA	Quando cliccato torna alla schermata M13 senza effettuare modifiche
Bottone	SALVA	Quando cliccato torna alla schermata M13 modificando l'email dell'account



2.6.16 Schermata di cambio password per i dipendenti al primo accesso



M16: Schermata cambio password dipendenti

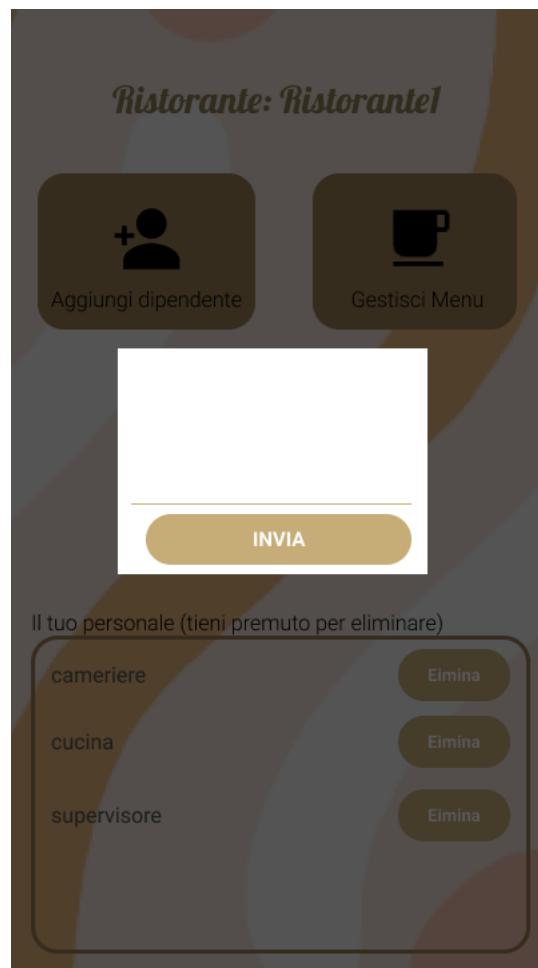
ID **M16**

Componenti:

Tipo	Nome	Funzione
Edit Text	NUOVA PASSWORD	Permette di inserire la nuova password per l'account del dipendente
Bottone	ANNULLA	Quando cliccato torna alla schermata M02 senza effettuare modifiche
Bottone	SALVA	Quando cliccato porta alla schermata M09 se è supervisore, M11 se è cameriere, M12 se è un account di cucina, modificando l'email dell'account



2.6.17 Schermata di creazione degli avvisi



M17: Schermata creazione avvisi

ID **M17**

Componenti:

Tipo	Nome	Funzione
Edit Text	AVVISO	Permette di inserire il testo dell'avviso
Bottone	INVIA	Quando cliccato torna alla schermata M06 inviando l'avviso



2.6.18 Schermata di aggiunta piatti al menù



M18: Schermata aggiunta piatti al menù

ID **M18**

Componenti:

Tabella 7:

Tipo	Nome	Funzione
EditText	NOME PRODOTTO	Permette di inserire il titolo del prodotto da inserire
EditText	DESCRIZIONE	Permette di inserire la descrizione del prodotto da Inserire
EditText	ALLERGENI	Permette di inserire gli allergeni contenuti nel prodotto

Continued on next page



Tabella 7: (Continued)

Tipo	Nome	Funzione
EditText	PREZZO	Permette di inserire il prezzo del prodotto
Spinner	TIPO	Permette di inserire il tipo di portata (antipasto, primo, ...)
Spinner	TIPO DI ALIMENTO	Permette di inserire il tipo di alimento (carne, pesce, ...)
Bottone	CERCA	Quando cliccato, ricerca il prodotto scritto nel database di OpenFoodFacts
Bottone	OK	Quando cliccato, ritorna alla schermata M08 salvando nel menu il prodotto
Bottone	ANNULLA	Quando cliccate, ritorna alla schermata M08 senza salvare il prodotto
Selettore	PRECONFEZIONATO	Se selezionato, indica che l'oggetto è di tipo preconfezionato(bibita, ...)



2.6.19 Schermata di aggiunta ordini



M19: Schermata aggiunta ordini

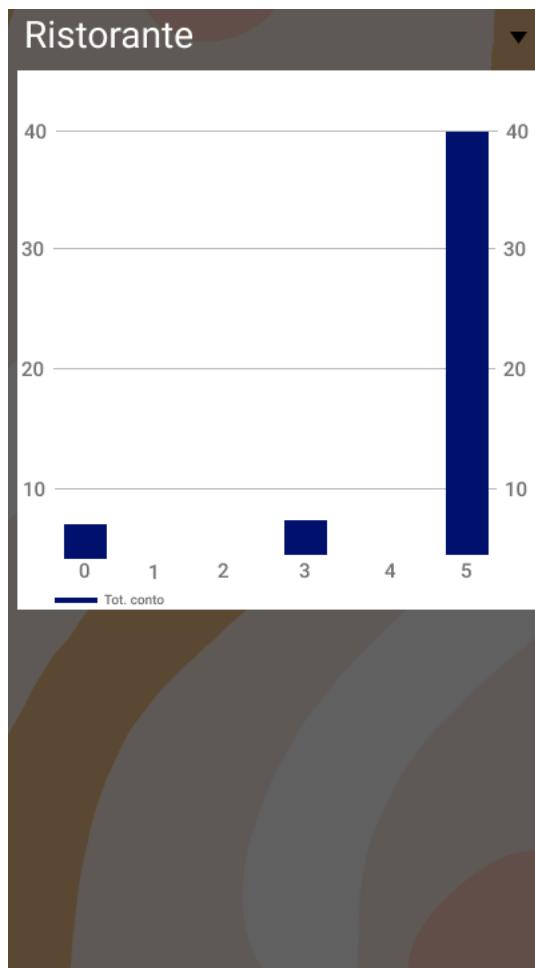
ID **M19**

Componenti:

Tipo	Nome	Funzione
ScrollView	PIATTI	Visualizza la lista dei piatti del menù del ristorante, dove ogni piatto si può cliccare per aggiungerlo all'ordine
Bottone	SALVA	Quando cliccato porta alla schermata M11 inviando l'ordine
Bottone	INDIETRO	Quando cliccato porta alla schermata M11 senza effettuare nessuna azione



2.6.20 Schermata di visualizzazione delle statistiche



M20: Schermata visualizzazione statistiche

ID **M20**

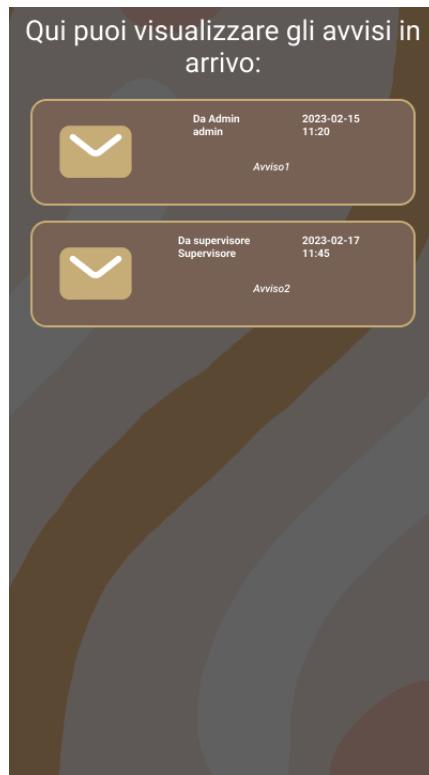
Componenti:

Tabella 8:

Tipologia	Nome	Funzione
Spinner	RISTORANTE	Permette di selezionare il ristorante di cui si vogliono visualizzare le statistiche
ScrollView	GRAFICI STATISTICHE	Permette di visualizzare i grafici relativi alle statistiche di <i>Ristorante</i>



2.6.21 Schermata di visualizzazione delle notifiche



M21: Schermata visualizzazione notifiche

ID M21

Componenti:

Tabella 9:

Tipologia	Nome	Funzione
ScrollView	NOTIFICHE	Permette di visualizzare le varie notifiche ricevute dal proprio ristorante, inviate dall'admin o dal supervisore, contenenti: <i>Nome, Data e ora, Messaggio</i> , che tramite swipe verso sinistra vengono segnati come letti ed eliminati dalla schermata



2.7 Valutazione dell'usabilità (a priori)

Per la valutazione dell'usabilità del nostro applicativo a priori, cioè prima della fase di sviluppo vera a propria, abbiamo deciso di imporci come linee guida le euristiche di Nielsen. Ne sono 10, ma vorremmo richiamare l'attenzione su alcune di esse nello specifico:

- 1. *Visibilità dello stato del sistema.* L'interfaccia presenta una discreta mancanza di feedback, che prevediamo di colmare con elementi quali Dialog, Toast, AlertDialog e Snackbar.
- 2. *Corrispondenza fra il mondo reale e il sistema.* Il sistema parla il linguaggio dell' utente utilizzatore del sistema. Tutto ciò che è inerente alle attività di ristorazione è riportato 1:1 in-app.
- 3. *Libertà e controllo da parte degli utenti.* In questo tipo di applicazione è raro che l'utente abbia bisogno di tasti undo e redo. In ogni caso, per funzioni complesse, saranno presenti dialog di conferma per eventualmente annullare determinate azioni.
- 4. *Consistenza e standard.* Ogni azione è ben contraddistinta da icone, segni, e flow che richiamano esattamente ciò che deve fare.
- 5. *Flessibilità ed efficienza d'uso* Riteniamo che il sistema da costruire non sia particolarmente difficile da gestire. Inoltre, le azioni sono sempre semplici e ben definite.
- 6. *Design minimalista ed estetico.* All'inizio abbiamo notato un design antiestetico ma semplice. Sarà affinato successivamente.
- 7. *Prevenzione degli errori.* Il sistema reagisce in maniera controllata e predeterminata alle situazioni di errori che gli utenti possono causare. Nulla è lasciato al caso, ed è, nelle build private dal team, gestita qualsiasi azione eseguibile dagli utenti.
- 8. *Riconoscere piuttosto che ricordare.* La nostra app è dotata di sezioni ben distinte, interfacce dinamiche a seconda del tipo di utente che le usa, e icone e testi esplicativi dell'azione che si va a intraprendere.
- 9. *Aiutare gli utenti a riconoscere gli errori, diagnosticarli e correggerli* Sono svariati i feedback sui campi di testo, i Toast, e i dialog se si cercano di compiere azioni illegali. Verranno poi migliorati.
- 10. *Guida e documentazione.* E' presente un simpatico topolino (che richiama il logo dell'app) che consiglia tramite vignette e piccoli dialoghi le azioni che si possono intraprendere. Inoltre ci sono piccole note che segnalano campi obbligatori e altri limiti.
Riteniamo che l'applicazione non necessiti un "manuale d'uso", in quanto, come già detto, è minimale e semplice da utilizzare.
L'applicativo è inoltre distribuito in ben tre lingue: italiano, inglese e spagnolo. Queste dovrebbero essere sufficienti a garantire la fruibilità a buona parte degli utenti italiani.

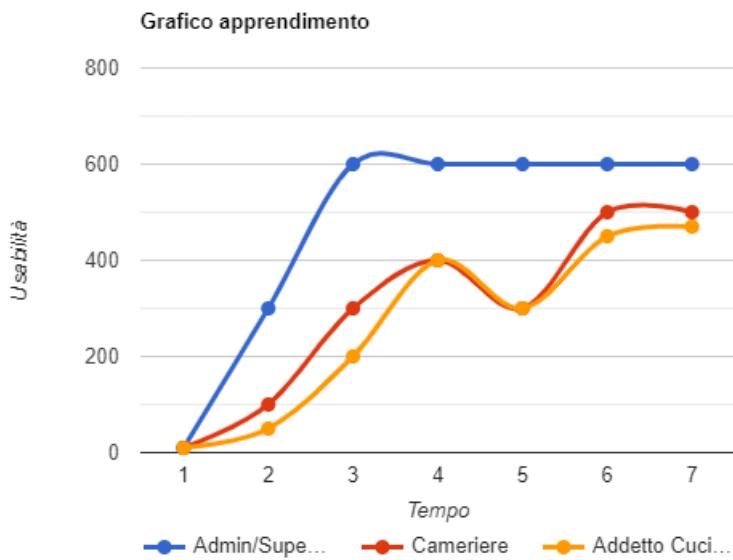


Figura 2.18: **Grafico:** Usabilità

Considerazioni sull'usabilità

Riteniamo, in base ad una prima analisi, che quelli che faranno più fatica ad apprendere il funzionamento dell'app potrebbero essere inizialmente admin e supervisori. Questo perchè, sono, nella nostra "gerarchia" di funzionalità, quelli che ne posseggono di più. Ma come possiamo notare, sono anche quelli capaci di mantenere costante il grado di conoscenza dell'applicativo. Camerieri e addetti alla cucina, in base anche ai turni svolti nel ristorante, alle fasce di punta più affolatte, potrebbero talvolta soffrire di rallentamento grafici, di bug, e di layout ancora non troppo ottimizzati per gestire grandi quantità di ordini. Inoltre sono il tipo di personale che più velocemente cambia in un ristorante, dovendo così riniziare il ciclo di apprendimento.



2.8 Individuazione target d'utenti

La conoscenza dell'utente finale è di importanza fondamentale per chi progetta sistemi software di questo tipo. La grande diversità degli esseri umani fa sì che, anche considerando compiti e contesti d'uso simili, un oggetto potrebbe risultare usabile per un certo utente e del tutto inusabile per un altro. Sicuramente, in base alle richieste del committente abbiamo subito individuato 4 principali categorie di utenti utilizzatori dell'app:

Admin: Amministratore e proprietario del ristorante. Una persona che deve avere tutto sotto controllo, può gestire le sue attività nonché i dipendenti che ne fanno parte, aggiungerne di nuovi e talvolta, purtroppo, eliminarli. Solitamente possiede più di un ristorante e deve essere in grado di gestire e comunicare con tutti i dipendenti facilmente.

Supervisore: Dopo l'amministratore, è, nella "gerarchia" da noi definita, la seconda persona con più funzioni disponibili in-app. Anch'esso dispone di una dashboard completa sullo stile dell'admin.

Cameriere/Addetto sala: Senza la figura del cameriere un'attività di ristorazione non va avanti. Essendo una figura spesso in cambiamento all'interno delle attività, sappiamo quant'è importante fornire a questi ultimi un applicativo funzionale, facile da usare e da apprendere: per questo la sua interfaccia è ottimizzata per un palmare o smartphone compatto.

Addetto Cucina: Riceve tutti gli ordini dai camerieri e li inoltra alla cucina. Anche lui dispone di un'interfaccia semplice e dinamica che perfettamente si adatta al suo ruolo nell'attività.

Tutto ciò è reso possibile da uno sviluppo che va incontro alle esigenze dei diversi utenti. La nostra interfaccia riconosce il tipo di utente che logga e cambia in base alle sue esigenze.

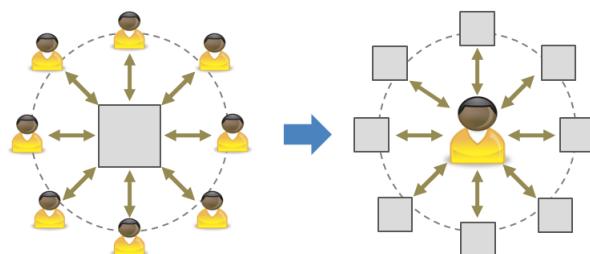


Figura: Da una visione centrata sul sistema a una visione centrata sull'utente



2.9 Glossario

In questa sezione vengono chiariti alcuni termini usati all'interno della documentazione, per rendere la lettura accessibile anche ai non esperti del settore.

- *RecyclerView*: potente approccio nella risoluzione di un problema comune: la creazione di liste per la visualizzazione (su Android) di dati ottenuti da un servizio remoto o da un database locale.
- *Adapter*: Un oggetto di tipo adapter in Android rappresenta un ponte tra un' AdapterView e i dati che questa deve rappresentare.
- *Dialog*: Tipo di popup che il sistema Android mette a disposizione che mostra una finestra di dialog personalizzabile.
- *Spring Boot*: Spring è uno strumento che permette a noi programmati di usare il linguaggio di programmazione ad oggetti java per scrivere ottime app lato server.
- *JPA*: Le Java Persistence API, talvolta riferite come JPA, sono un framework per il linguaggio di programmazione Java che si occupa della gestione della persistenza dei dati di un DBMS relazionale
- *MVC*: Pattern Model-View-Controller.
- *Three-Tier Architecture*: l'espressione architettura three-tier ("a tre strati") indica una particolare architettura software e hardware di tipo multi-tier per l'esecuzione di un'applicazione web che prevede la suddivisione dell'applicazione in tre diversi moduli o strati dedicati rispettivamente alla interfaccia utente, alla logica funzionale (business logic) e alla gestione dei dati persistenti. Android e Spring-Boot ne supportano i principi.
- *HTTP*: Protocollo di comunicazione client-server
- *OpenFoodFacts*: Database online che raccoglie le informazioni(allergeni, ingredienti, ecc.) di molti cibi preconfezionati.



2.10 Class diagram di analisi o dominio

Class diagram a livello concettuale Studia i concetti propri del dominio sotto studio, senza preoccuparsi della loro successiva implementazione. È una sorta di Dizionario Visuale del dominio.

Class diagram di specifica implementativa Rappresenta la struttura implementativa, specificando come va sviluppato il sistema. È un raffinamento del precedente.

I Class diagram di analisi o di dominio ci forniscono un'indicazione iniziale sulle componenti principali del software a prescindere dall'architettura e dalle tecnologie che verranno poi scelte per implementarli. In questa fase infatti ancora di analisi, il team sta concretizzando i requisiti in qualcosa di pratico da dare in pasto ai programmati, che ne seguiranno il modello per arrivare al prodotto finito.

2.10.1 Class diagram Login

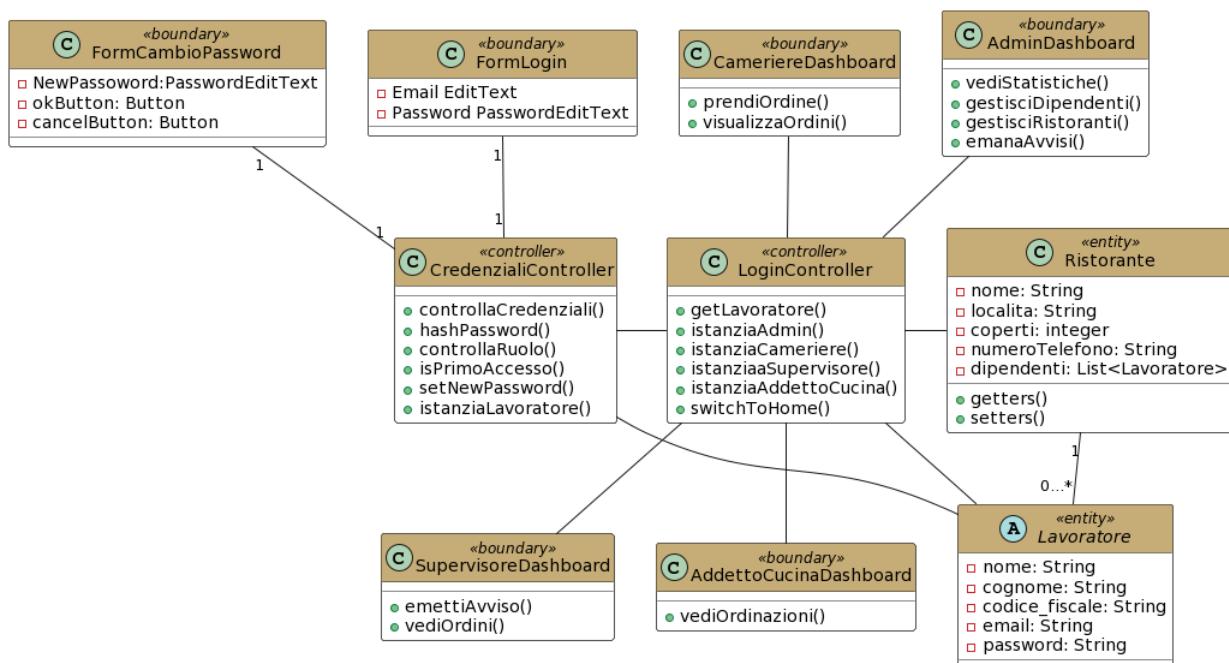


Figura 2.19: C01: Class diagram Login



2.10.2 Class diagram Ristorante

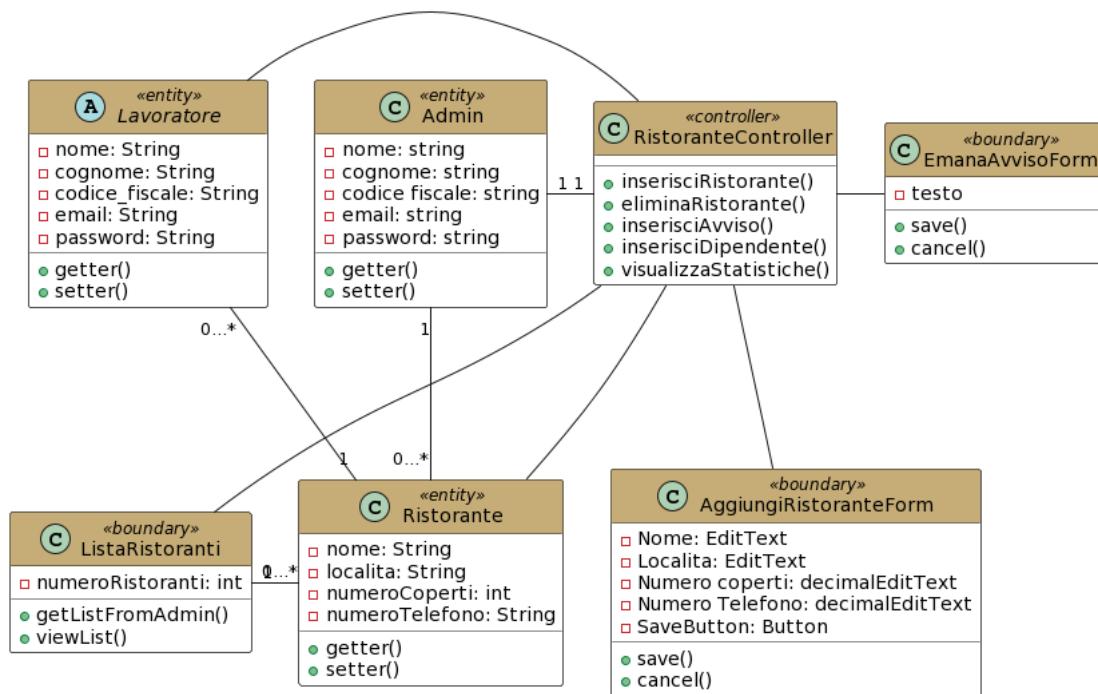


Figura 2.20: C02: Class diagram Gestione ristorante



2.10.3 Class diagram dipendenti

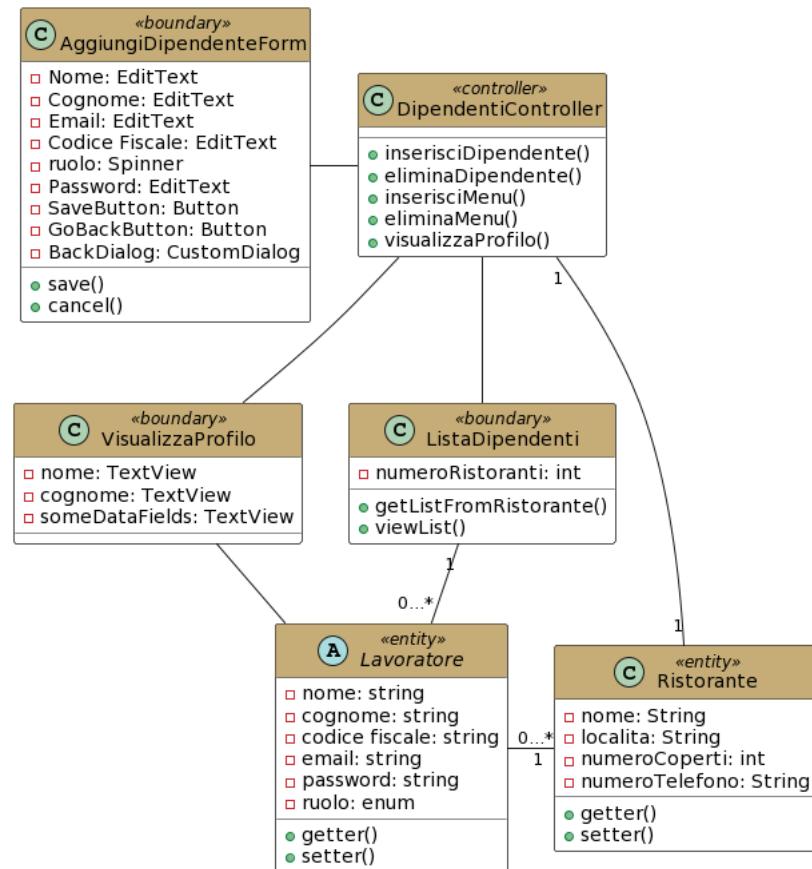


Figura 2.21: C03: Class diagram gestione dipendenti



2.10.4 Class diagram Menu

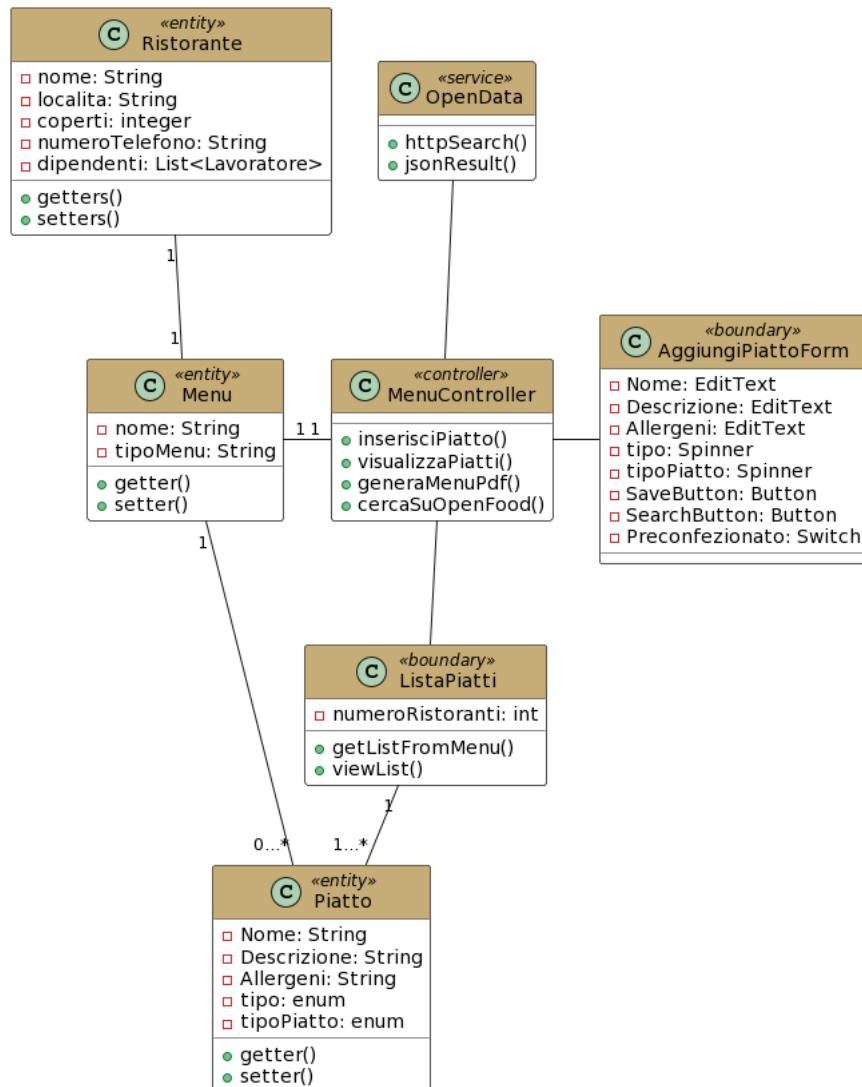


Figura 2.22: C04: Class diagram gestione piatti



2.10.5 Class diagram Avvisi

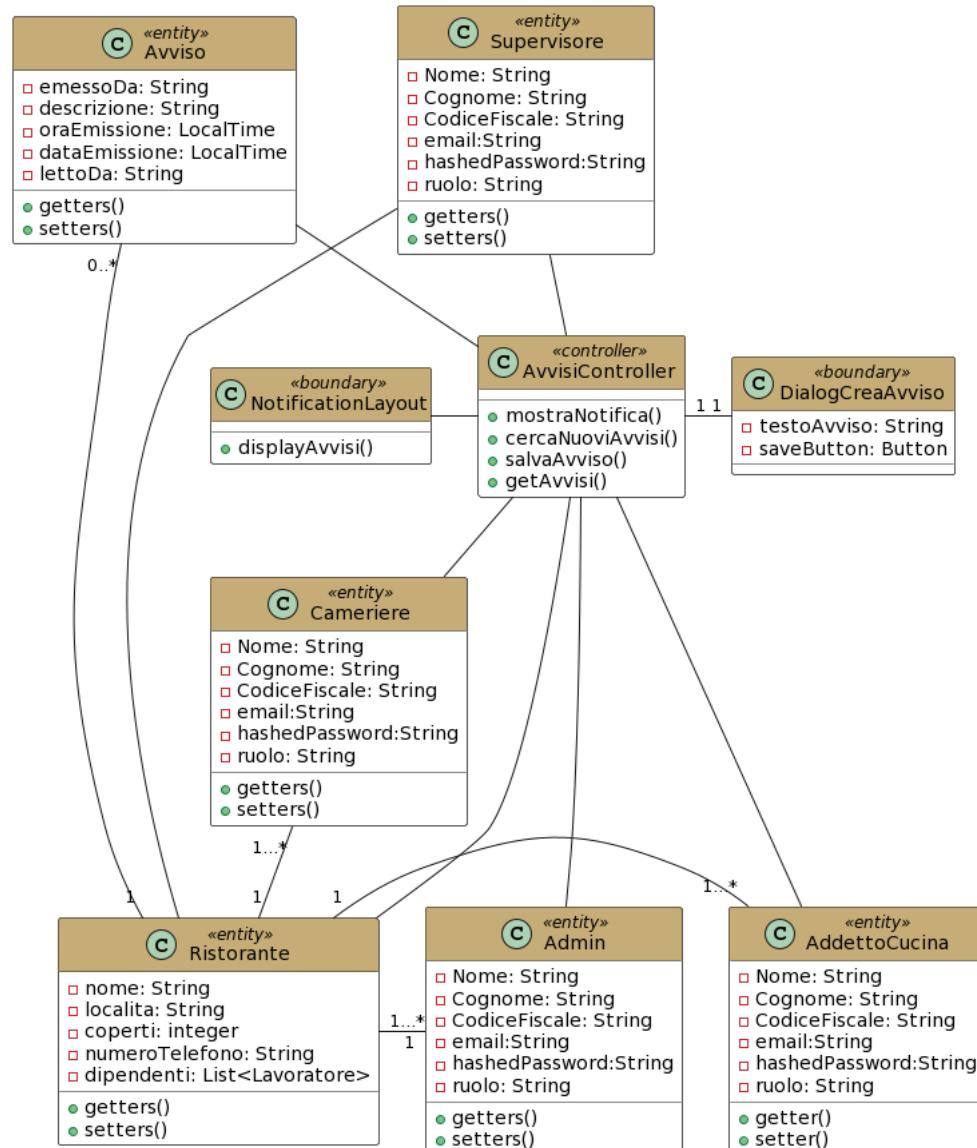


Figura 2.23: C05: Class diagram gestione avvisi



2.10.6 Class diagram Ordini

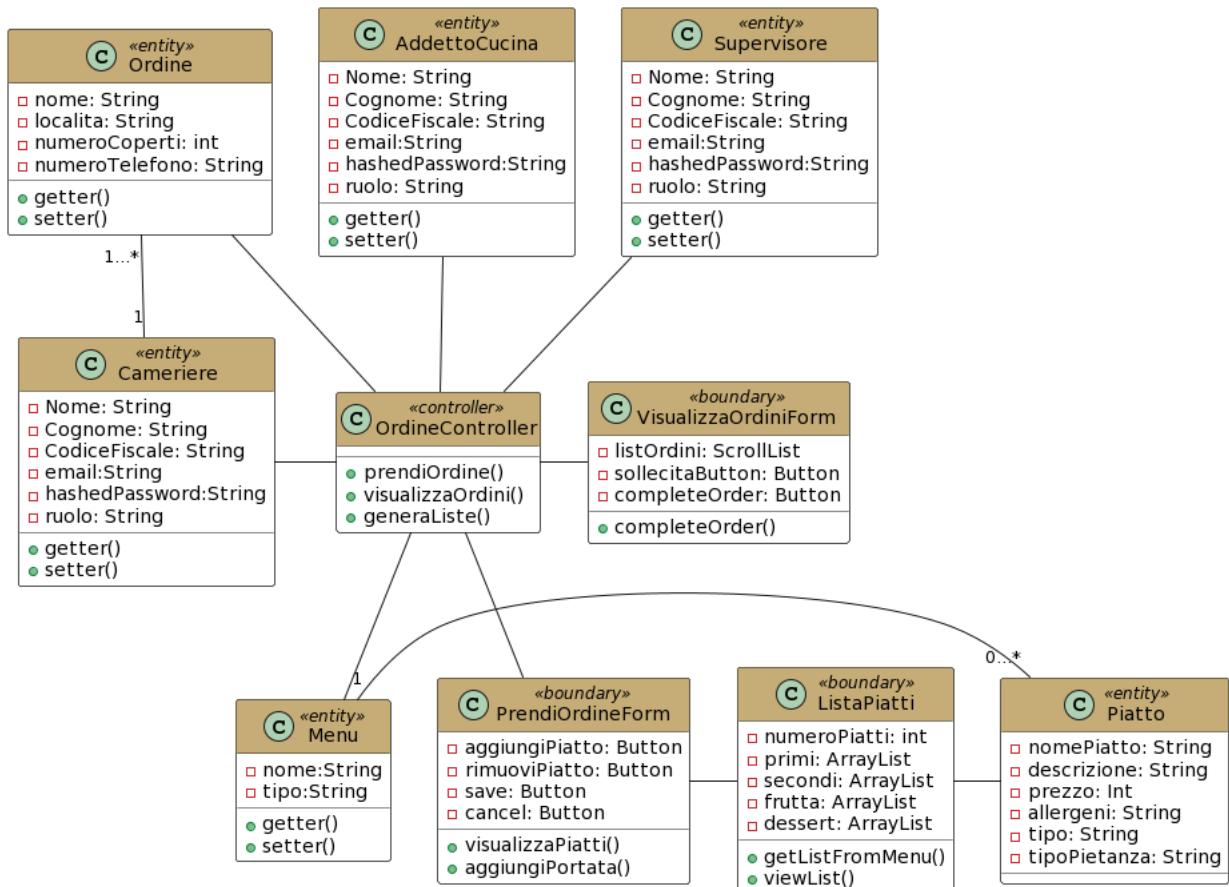


Figura 2.24: C06: Class diagram gestione ordini



2.10.7 Class diagram statistiche

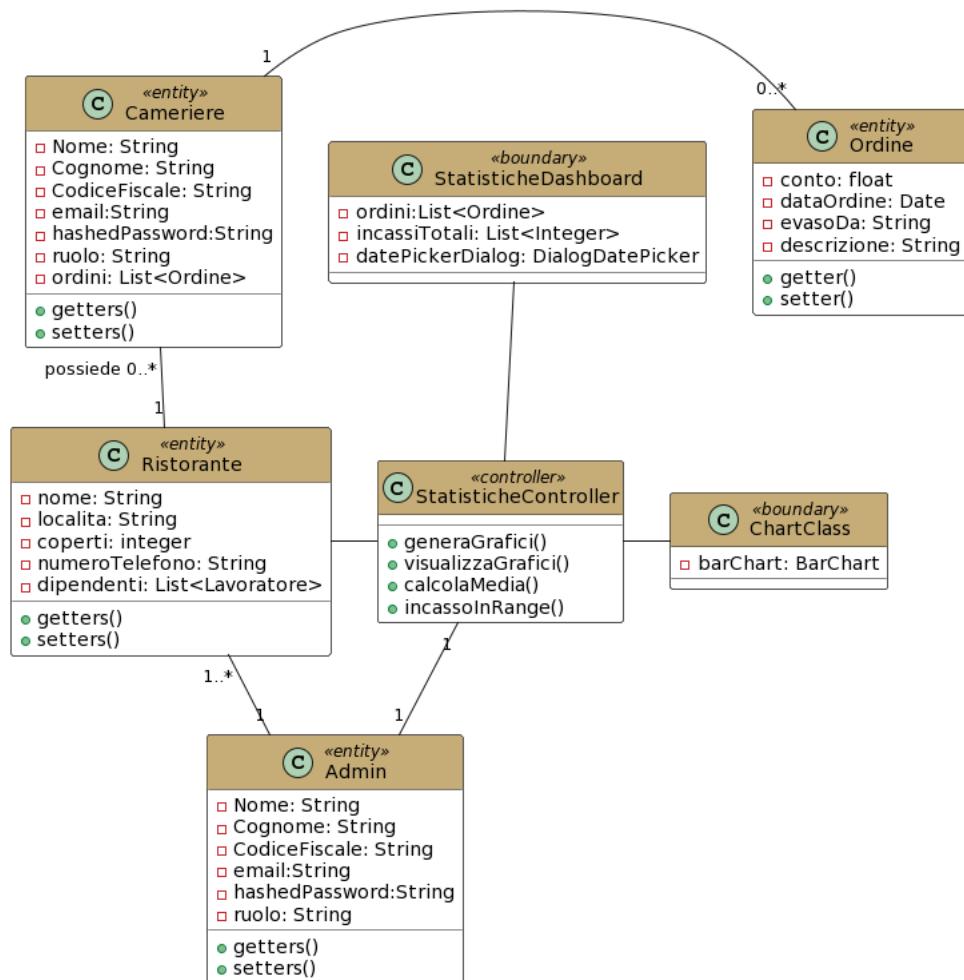


Figura 2.25: C07: Class diagram gestione statistiche



2.11 Sequence di analisi

Abbiamo scelto di modellare (in fase di analisi dei requisiti) con i sequence diagram i seguenti casi d'uso:

- Aggiunta menu del ristorante
- Prendere ordinazione

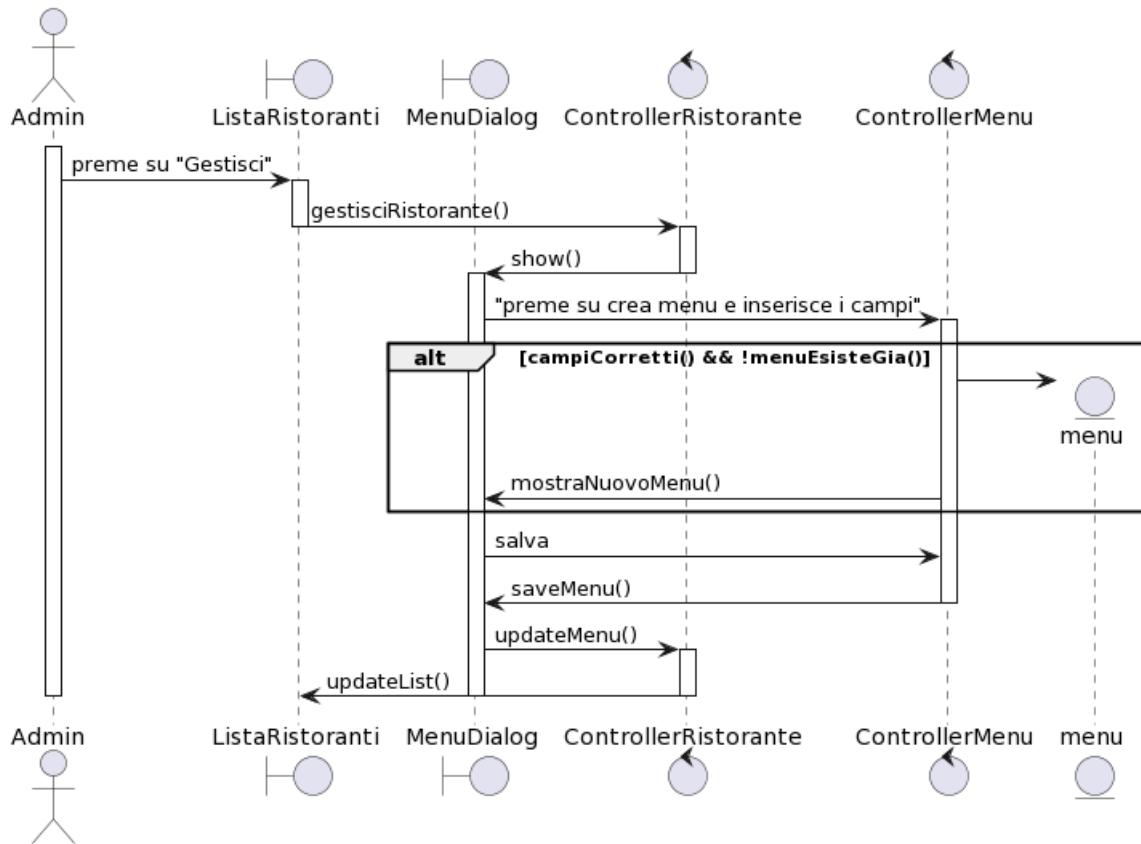


Figura 2.26: Sequence: Aggiungi menu

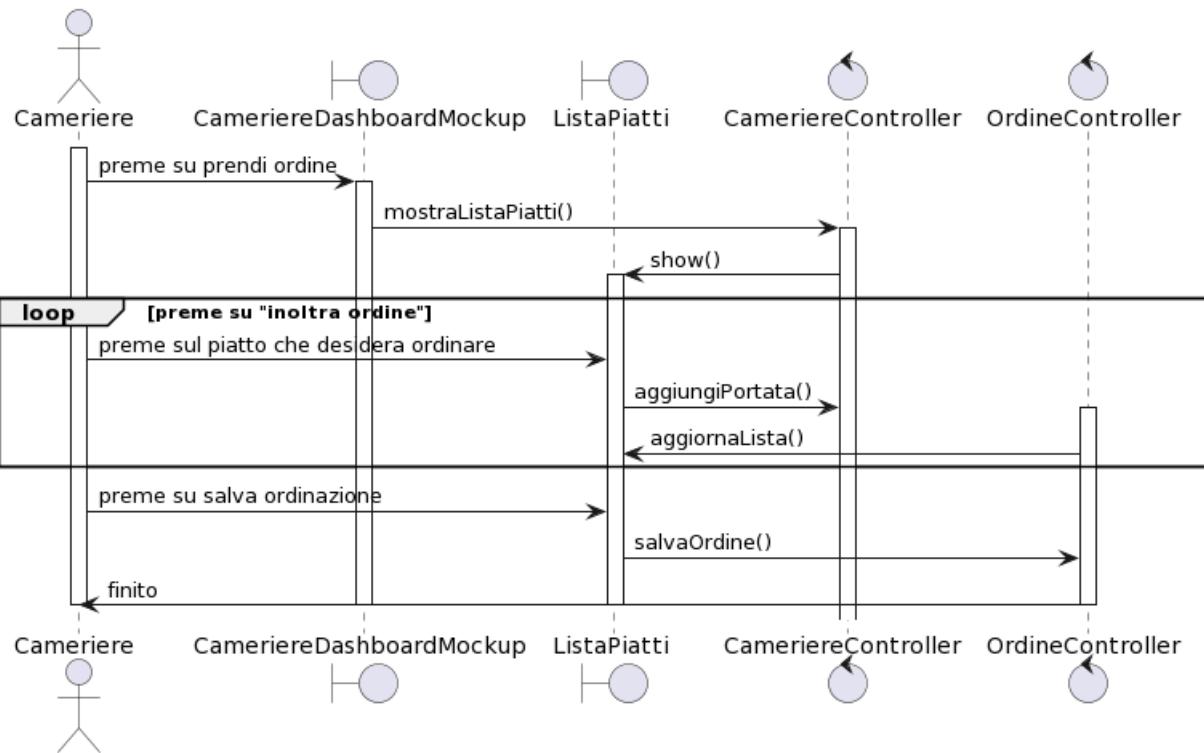


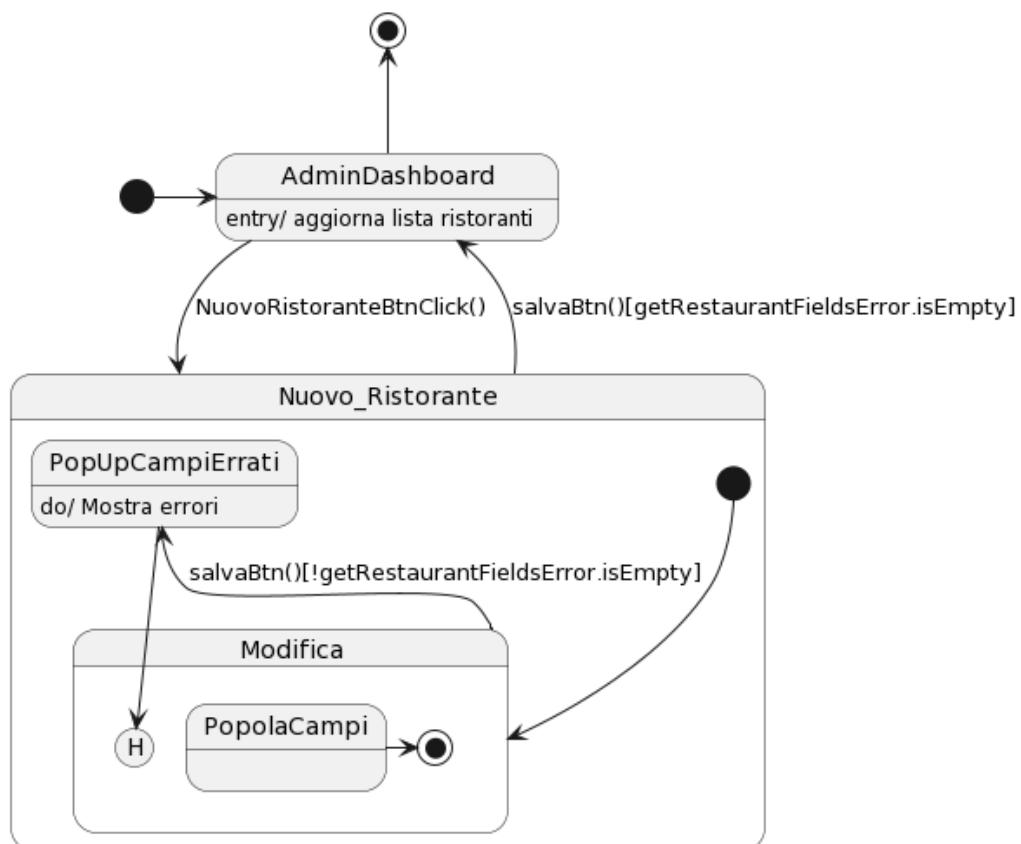
Figura 2.27: Sequence: Prendi ordine



2.12 Prototipazione funzionale via statechart dell'interfaccia grafica

Gli statechart sono uno strumento di modellazione grafica, utile per rappresentare il funzionamento di un sistema. Sono caratterizzati da stati e transizioni, che rappresentano il passaggio tra stati diversi. Sono inoltre molto importanti poiché forniscono un chiaro mezzo di costruzione di prototipi statici: con pochi semplici simboli è possibile modellare l'intero funzionamento di un'interfaccia grafica. Useremo questo potente strumento per rappresentare i seguenti casi d'uso: *Aggiungi ristorante*, *Aggiungi piatto*, *Prendi ordine* e *Visualizza avvisi*.

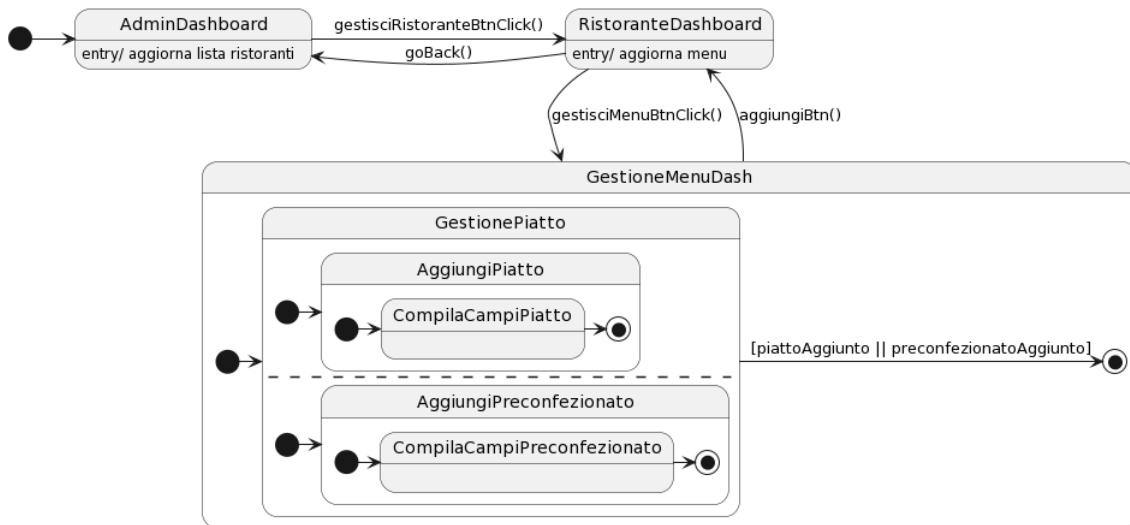
2.12.1 Aggiungi ristorante



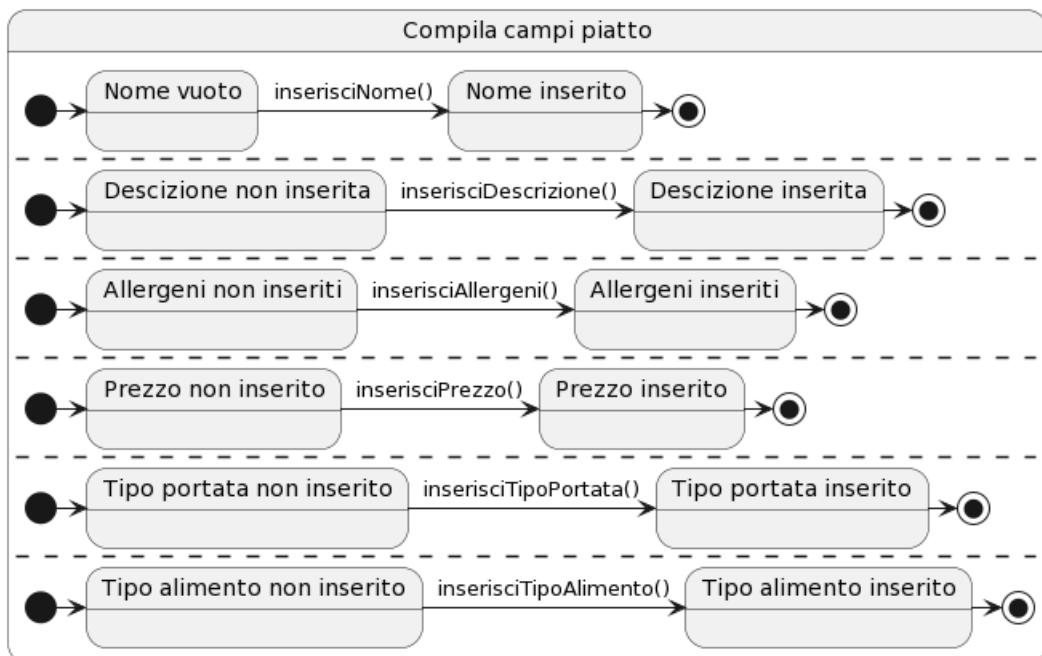
Statechart : aggiunta ristorante



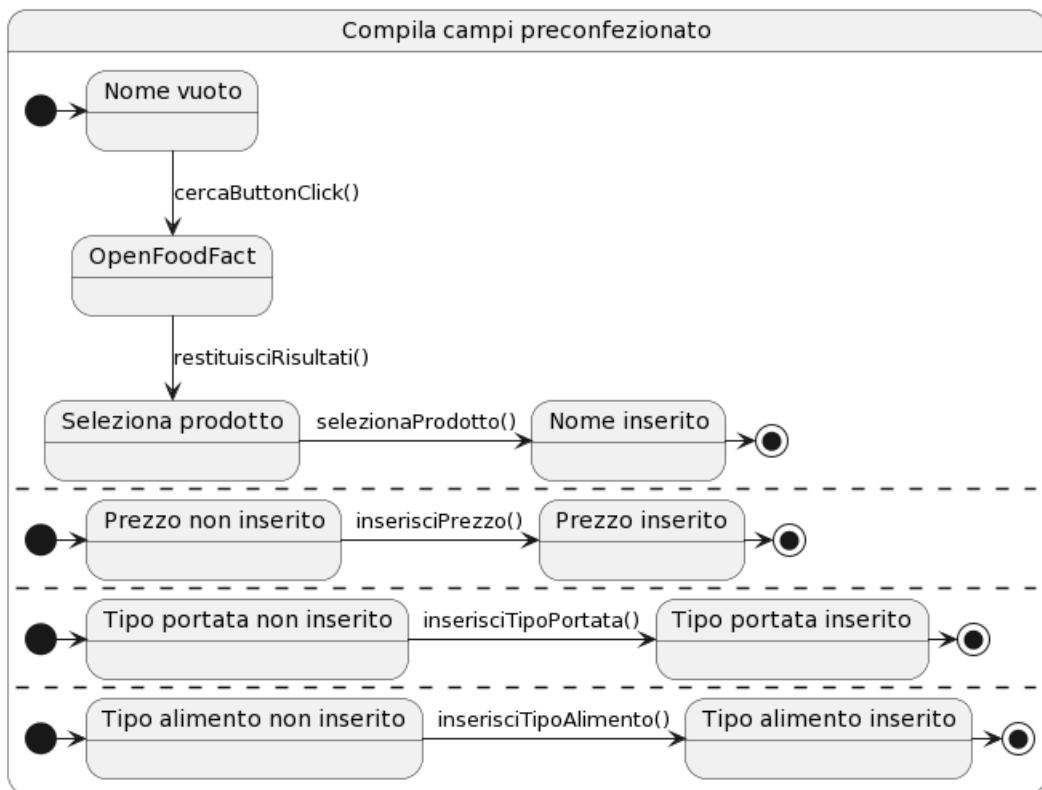
2.12.2 Aggiungi piatto



Statechart : aggiunta piatto

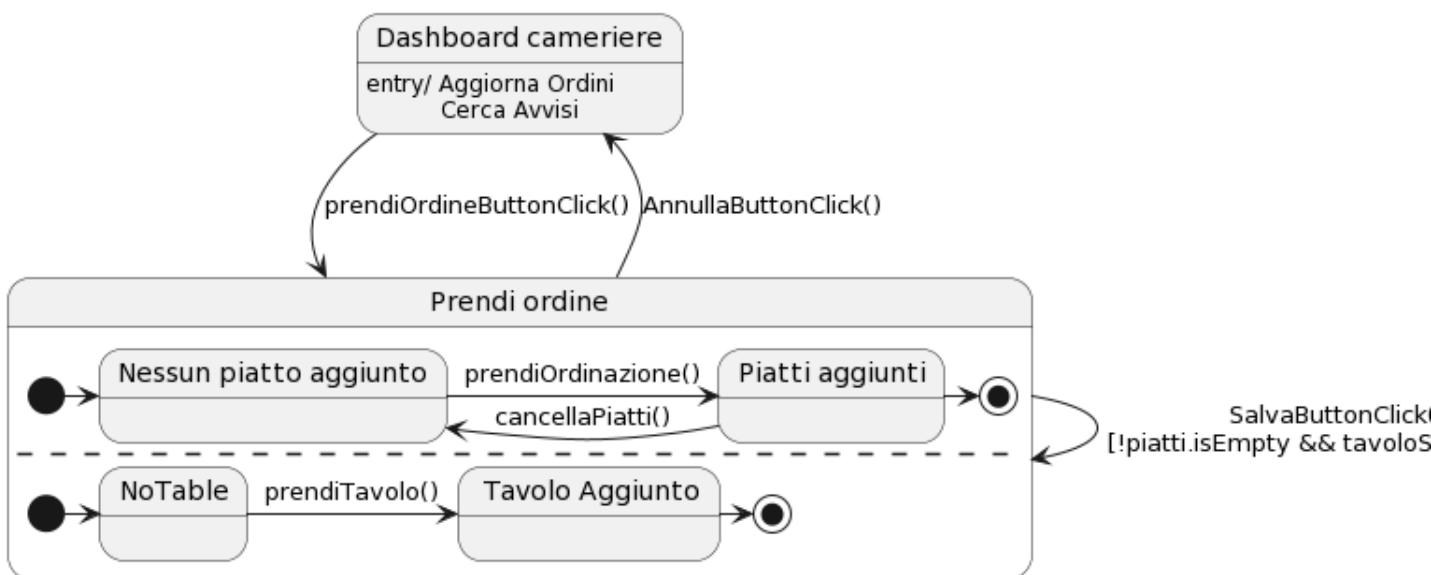


Statechart : compilazione campi piatto non preconfezionato



Statechart : compilazione campi piatto preconfezionato

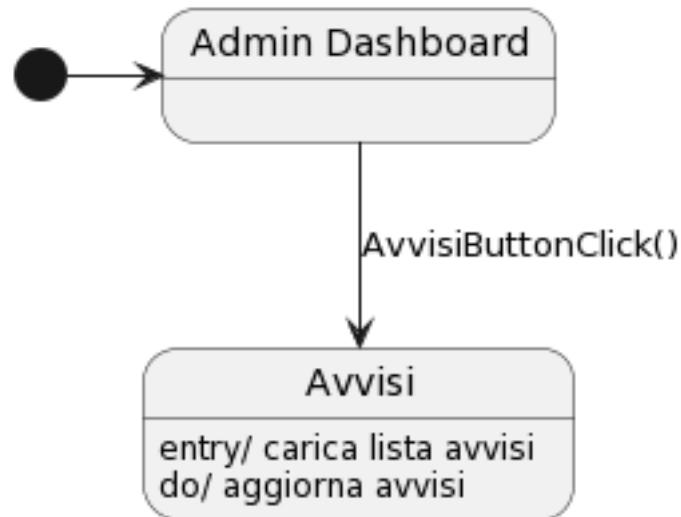
2.12.3 Prendi ordine



Statechart : Prendi ordinazione



2.12.4 Visualizza avvisi



Statechart : visualizzazione avvisi



3 Documento di design

3.1 Architettura del sistema

Alla luce delle valutazioni effettuate durante l'analisi dei requisiti, abbiamo trovato conveniente strutturare la nostra app seguendo un tipo di architettura generale client-server (con server centralizzato). I client, nel nostro caso, sono quasi sempre dei dispositivi portatili dotati di sistema operativo android (con la possibilità volendo di utilizzare emulatori da pc desktop) mentre il server è stato realizzato con il framework di sviluppo Spring Boot. Il client comunica con il server che offre delle REST API attraverso delle richieste HTTP (le risposte del server saranno quasi sempre in formato JSON).

3.1.1 Client

Come già detto, il client è composto da un'applicazione android distribuita o tramite apk o tramite play store (rispetta tutti i requisiti per essere pubblicata) che mira ad offrire un'interfaccia utente semplice e pulita per l'accesso agli endpoint forniti dal nostro server Spring-Boot.

In generale, l'architettura di un progetto Android di default segue il pattern architettonicale *Model-View-Controller (MVC)*, ma con alcune modifiche per adattarsi al contesto specifico di Android. In aggiunta, elenchiamo qui altri pattern e librerie che abbiamo adoperato per lo sviluppo dell'app:

Pattern Singleton: Abbiamo usato il pattern Singleton per generare una sola istanza del dipendente che si logga nell'app. Così facendo, è risultato più semplice aggiornare gli oggetti entità quali Cameriere, Addetto Cucina, Admin e Supervisore e mantenere costantemente un rapporto 1:1 con quanto salvato nel server.

Pattern Observer: Molte componenti del sistema Android utilizzano il pattern Observer. Il pattern Observer prevede che un oggetto, chiamato soggetto (Subject), mantenga una lista di oggetti dipendenti, chiamati osservatori (Observers), e che questi vengano notificati automaticamente ogni volta che lo stato del soggetto cambia. In questo modo, gli osservatori possono essere aggiornati in modo efficiente sullo stato del soggetto senza dover verificare continuamente se qualcosa è cambiato. In Android, il pattern Observer viene utilizzato, ad esempio, per gestire gli eventi generati dall'utente (come i tap sullo schermo), per notificare i cambiamenti nei dati o per gestire la comunicazione tra componenti diversi dell'applicazione. Anche l'implementazione di **CallBack** come nel nostro caso (con VolleyCallback) possono essere viste come un' implementazione del pattern Observer. Più in particolare la nostra VolleyCallback, che altro non è che un'interfaccia, richiede di implementare in ogni Activity il metodo onResponse() che aggiorna i dati locali con i dati ricevuti dal server. Ciò che andiamo ad aggiornare in locale sono poi i Singleton del lavoratore loggato. Da lì, si passa poi all'aggiornamento delle viste nel caso di novità nell'oggetto risposta ricevuto.

Volley Volley è una libreria Android sviluppata da Google che semplifica l'elaborazione di richieste di rete in applicazioni Android. È stata progettata per gestire facilmente le chiamate REST API e può essere utilizzata per inviare e ricevere dati in diversi formati, come JSON, XML e immagini. Volley utilizza una combinazione di caching e pooling di connessioni per garantire prestazioni elevate e ridurre il consumo di risorse. Inoltre, supporta la gestione automatica di thread e gestisce in modo trasparente gli errori di rete, semplificando notevolmente la scrittura di codice robusto e affidabile. Implementata come Singleton, garantisce l'esistenza di una sola istanza di essa, e ci assicura consumi di memoria costanti e limitati.



Gson Gson è una libreria Java sviluppata da Google che permette la serializzazione e la deserializzazione di oggetti JSON (JavaScript Object Notation) in oggetti Java. Gson offre una serie di metodi per la conversione di oggetti Java in JSON e viceversa, semplificando il processo di scambio di dati tra applicazioni Java e servizi web che utilizzano il formato JSON.

3.1.2 Server

Il back-end della nostra applicazione è stato realizzato con la tecnologia offerta da Spring Boot, un potente framework di sviluppo che sfrutta Java. L'architettura che prevede è anch'essa a 3 livelli e prevede i seguenti componenti:

- Model: L'insieme di classi che rappresentano le entità della nostra applicazione. Le stesse sono riportate sul client, e, cosa più importante, Spring mappa le classi con annotation "@entity" 1:1 con le tabelle nel db relazionale.
- Repository: Sfruttando il framework JPA (Java Persistence API) riusciamo a gestire persistenza e consistenza dei dati nel db in maniera quasi automatizzata. Spring ci fornisce un gran supporto in questo ambito.
- Controller: Qui ci va tutta la logica di controllo del server. Sono queste le classi che espongono gli end-point Rest ai client e, comunicando con le repository aggiornano i model e le tabelle nel db allo stesso tempo. Nel controller mappiamo tutti gli end-point a reindirizziamo le richieste http alla rispettiva funzione.

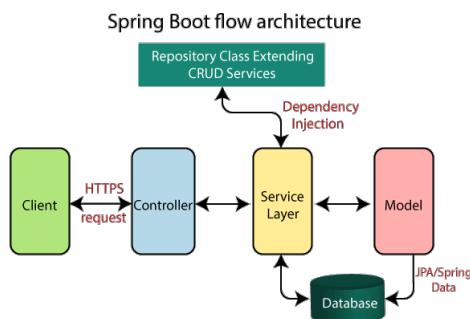


Figura: Spring-Boot

Perchè Spring?

La scelta di utilizzare Spring è dovuta principalmente alle sue caratteristiche di flessibilità e scalabilità. Semplifica inoltre lo sviluppo dell'intero applicativo grazie alle sue funzionalità di auto-configurazione e di gestione delle dipendenze, che avviene con Gradle e Maven. Spring Boot fornisce inoltre un'ampia gamma di funzionalità per sviluppare applicazioni web in modo rapido e facile. Queste funzionalità includono la gestione delle richieste HTTP, la gestione delle sessioni, la sicurezza, la gestione delle transazioni e, tra le altre, il pieno supporto alle API REST e operazioni CRUD.

In sintesi, Spring Boot semplifica lo sviluppo di servizi web basati su REST e consente di creare API performanti, scalabili e sicure in modo rapido ed efficiente.



Microsoft Azure

Microsoft Azure è la piattaforma cloud pubblica di Microsoft, che offre servizi di cloud computing. Tra i vari piani che mette a disposizione abbiamo usufruito di quello "gratuito" che ci ha permesso di eseguire una macchia virtuale linux (seppur con poche risorse) e un database relazionale, che nel nostro caso è stato PostgreSQL. La configurazione della macchina virtuale non ha richiesto particolari abilità: tramite connessione ssh abbiamo configurato il progetto Spring Boot e avviandolo mettiamo a disposizione gli end-point raggiungibili poi dai client. L'indirizzo del server è *20.86.153.84* e la porta predefinita per le richieste HTTP è la 8080, che accetterà solo determinate richieste. Tutte le altre opzioni di configurazione sono disponibili nel pannello di controllo Azure nel quale il team può facilmente accedere: sicurezza, firewall, potenza computazionale e nuovi servizi sono facilmente applicabili. La potenza computazionale in questa fase di sviluppo e di utilizzo limitato dell'applicativo potrebbe risultare esile, ma il vantaggio di piattaforme quali Azure è possibile in qualsiasi momento aumentare le risorse dedicate a db e/o macchina virtuale, aumentando di fatto la scalabilità della nostra applicazione.

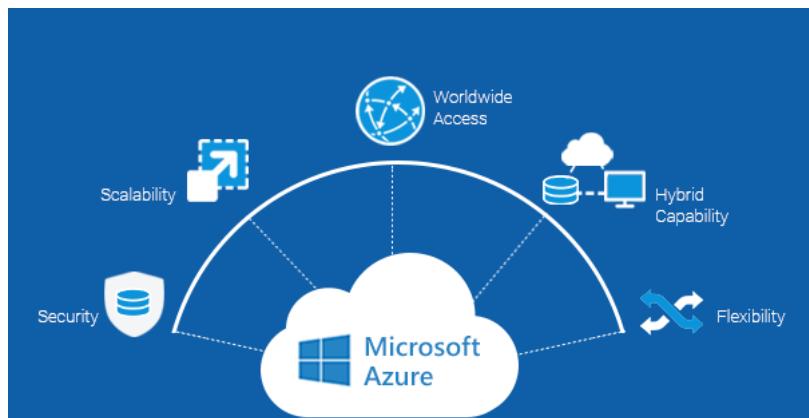


Figura: Microsoft Azure



E la sicurezza?

La sicurezza è garantita sia dalla sicurezza intrinseca dei servizi offerti da Microsoft Azure (parliamo quindi di operatività, assistenza, scalabilità) sia da un piccolo sistema di sicurezza pensato dal team:

```
@Component
public class AppCodeInterceptor implements HandlerInterceptor {

    private static final String HEADER_APP_CODE = "app_code";
    private static final String EXPECTED_APP_CODE = "Ratatouille23";

    @Override
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        String appCode = request.getHeader(HEADER_APP_CODE);
        if (appCode == null || !appCode.equals(EXPECTED_APP_CODE)) {
            System.out.println("Access_denied");
            response.setStatus(HttpStatus.UNAUTHORIZED.value());
            return false;
        }
        return true;
    }
}
```

Questa piccola parte di codice presente nel back-end, ci garantisce che nessun'altra richiesta al di fuori da quelle effettuate con il parametro "appcode" da noi definito sul client venga accettata. Estranei e malintenzionati, riceveranno lo status 401 se proveranno ad accedere a qualunque dei nostri endpoint.



3.2 Sequence diagram di design

3.2.1 Funzionalità: "Aggiungi Ristorante"

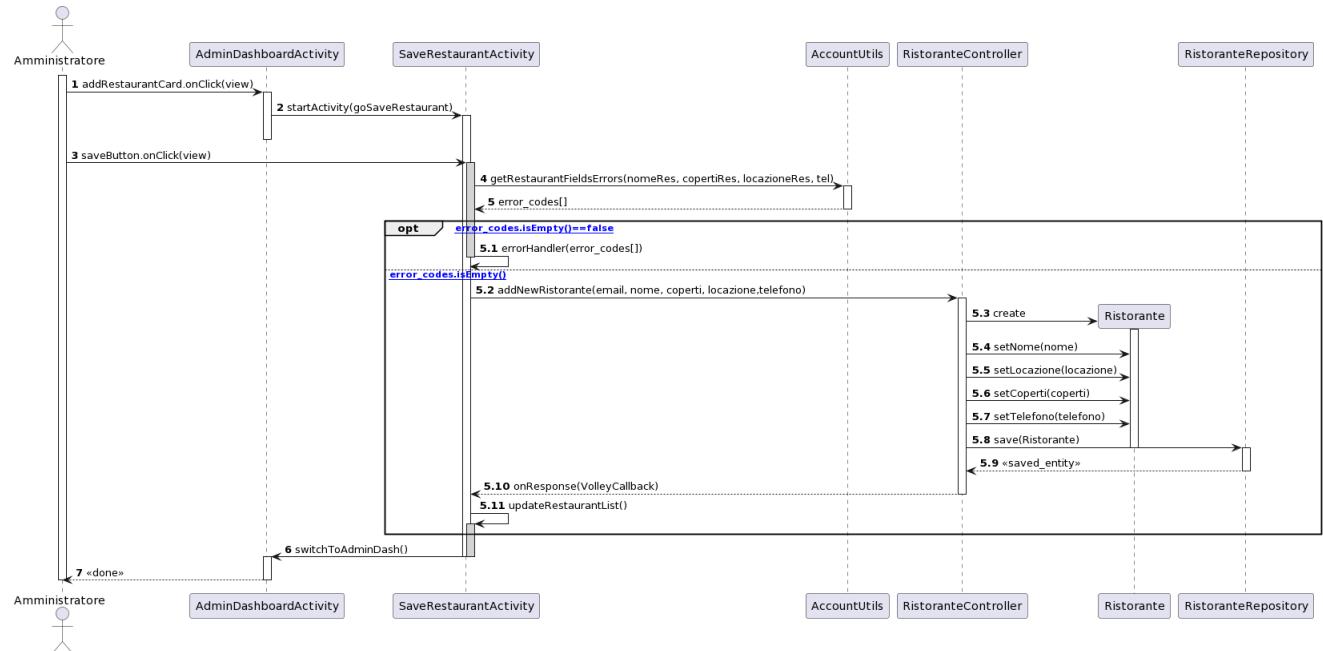


Figura 3.1: Sequence: Salva ristorante

3.2.2 Funzionalità: "Aggiungi portata"

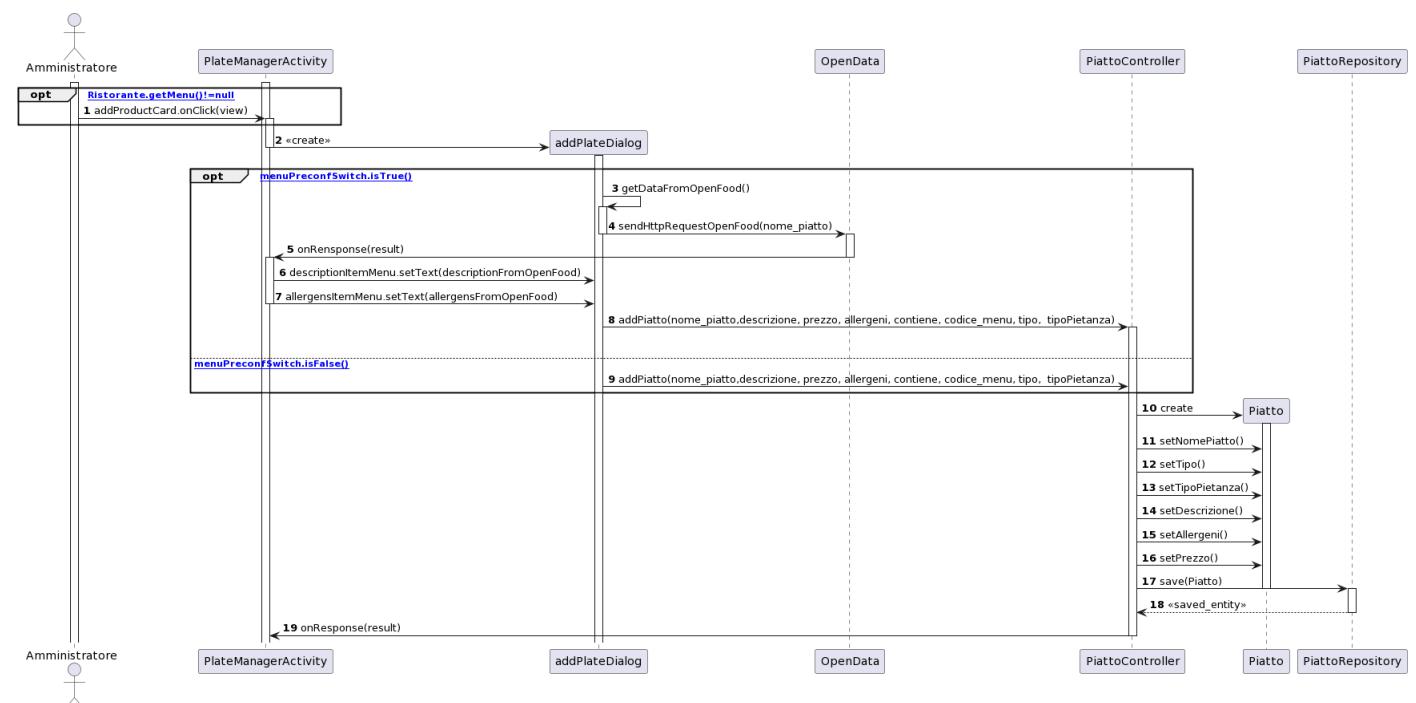


Figura 3.2: Sequence: Aggiungi portata



3.2.3 Funzionalità: "Prendi ordinazione"

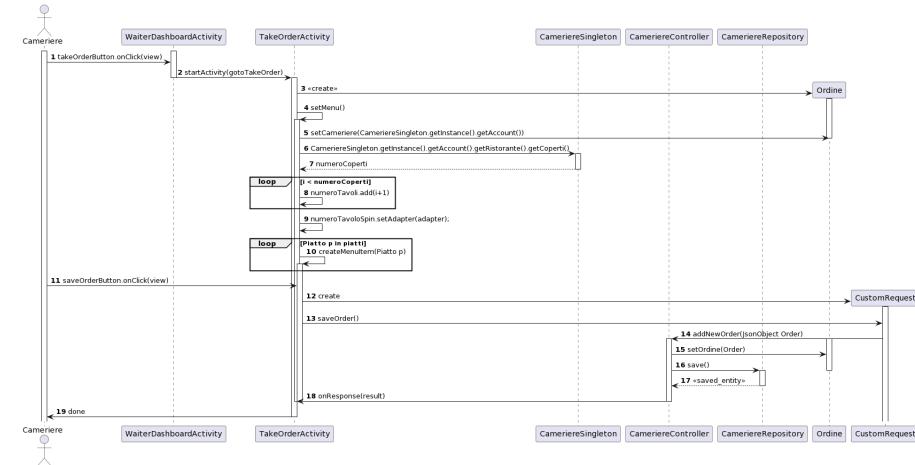


Figura 3.3: Sequence: Prendi Ordinazione

3.2.4 Funzionalità: "Visualizza avvisi"

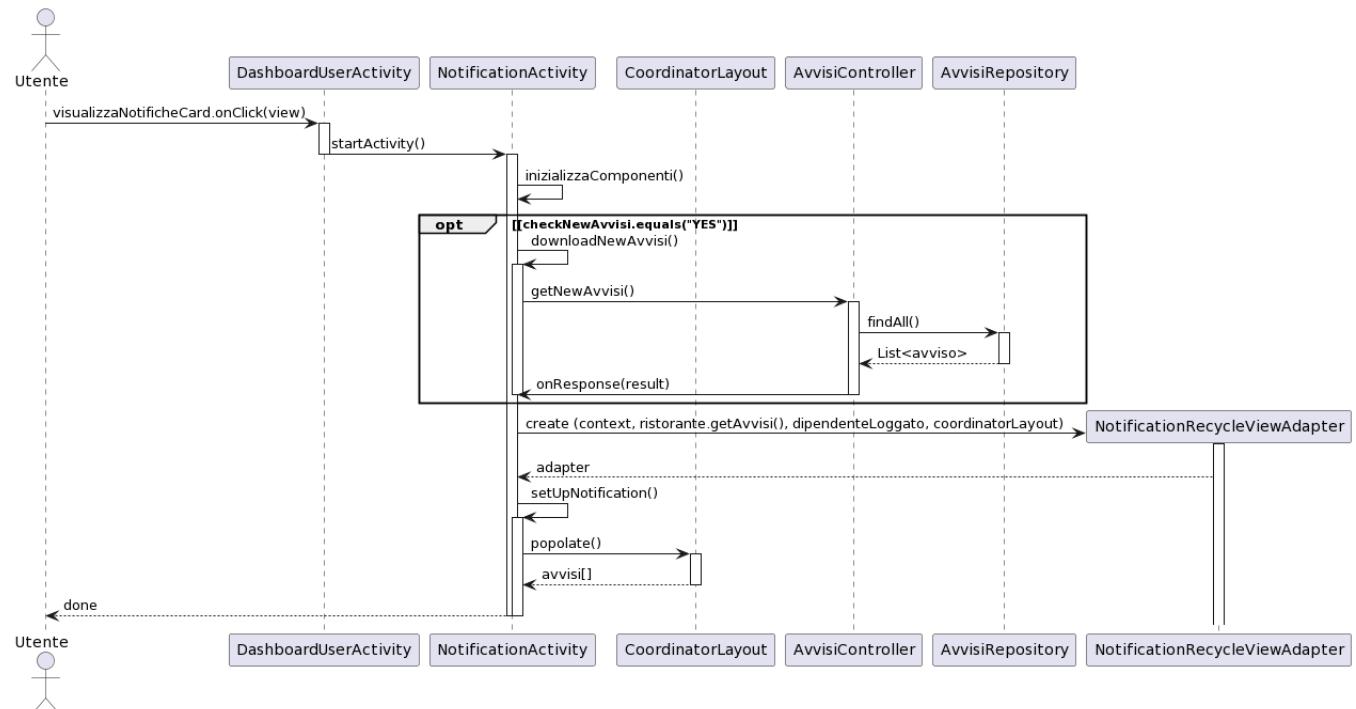


Figura 3.4: Sequence: Visualizza avvisi

3.3 Class diagram di design

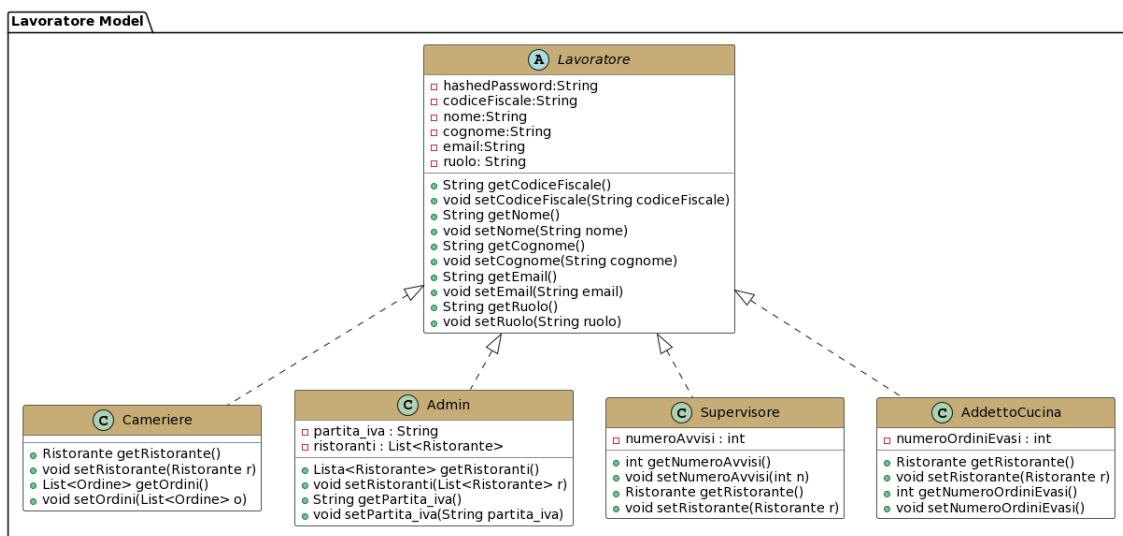
Verranno di seguito riportati i class diagram di design, che ora rispecchiano l'architettura del sistema e tutte le funzionalità nella loro interezza.

Si noti che per migliorare la leggibilità i diagrammi sono stati divisi in delle macrocategorie seguendo quelli che erano i modelli di dominio precedenti. Ogni diagramma è completo delle funzionalità descritte precedentemente negli use case e nei requisiti.

Note: Tutti i costruttori banali (senza argomenti) e i getter e i setter sono stati omessi per rendere più leggibili i diagrammi.

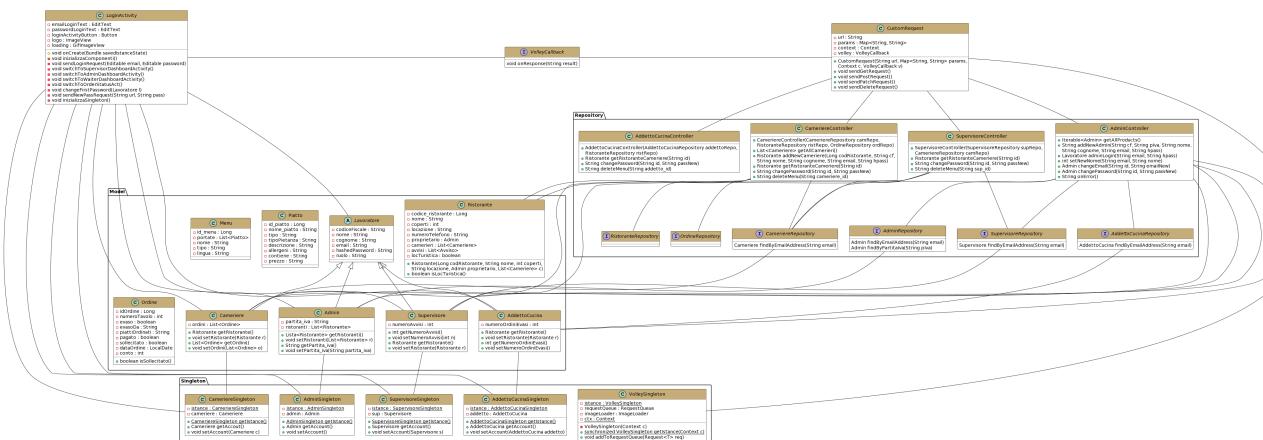
Nota 2: Le componenti grafiche (TextView, Button, ecc.) potrebbero non essere tutte presenti come attributi della classe, in quanto a volte sono stati dichiarati localmente nelle funzioni.

3.3.1 Espansione modelli Lavoratore



CD00: Class diagram Lavoratore

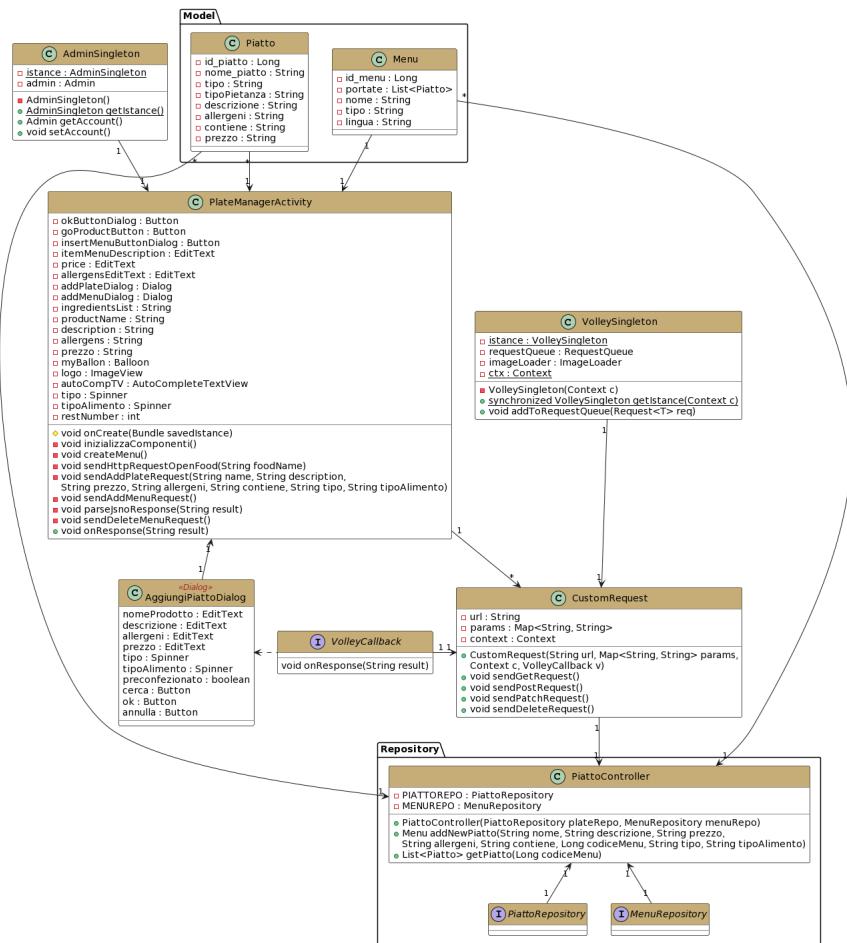
3.3.2 Login



CD01: Class diagram Login

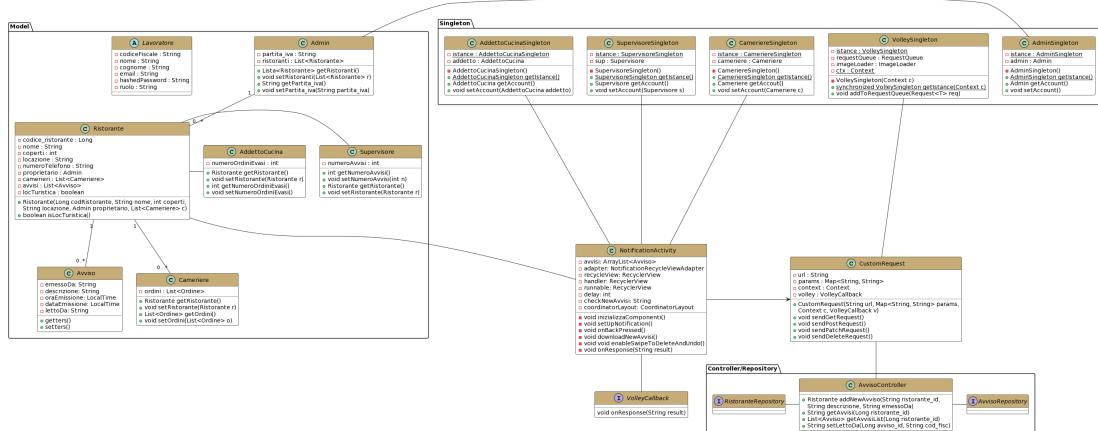


3.3.3 Gestione piatti



CD02: Class diagram gestione piatti

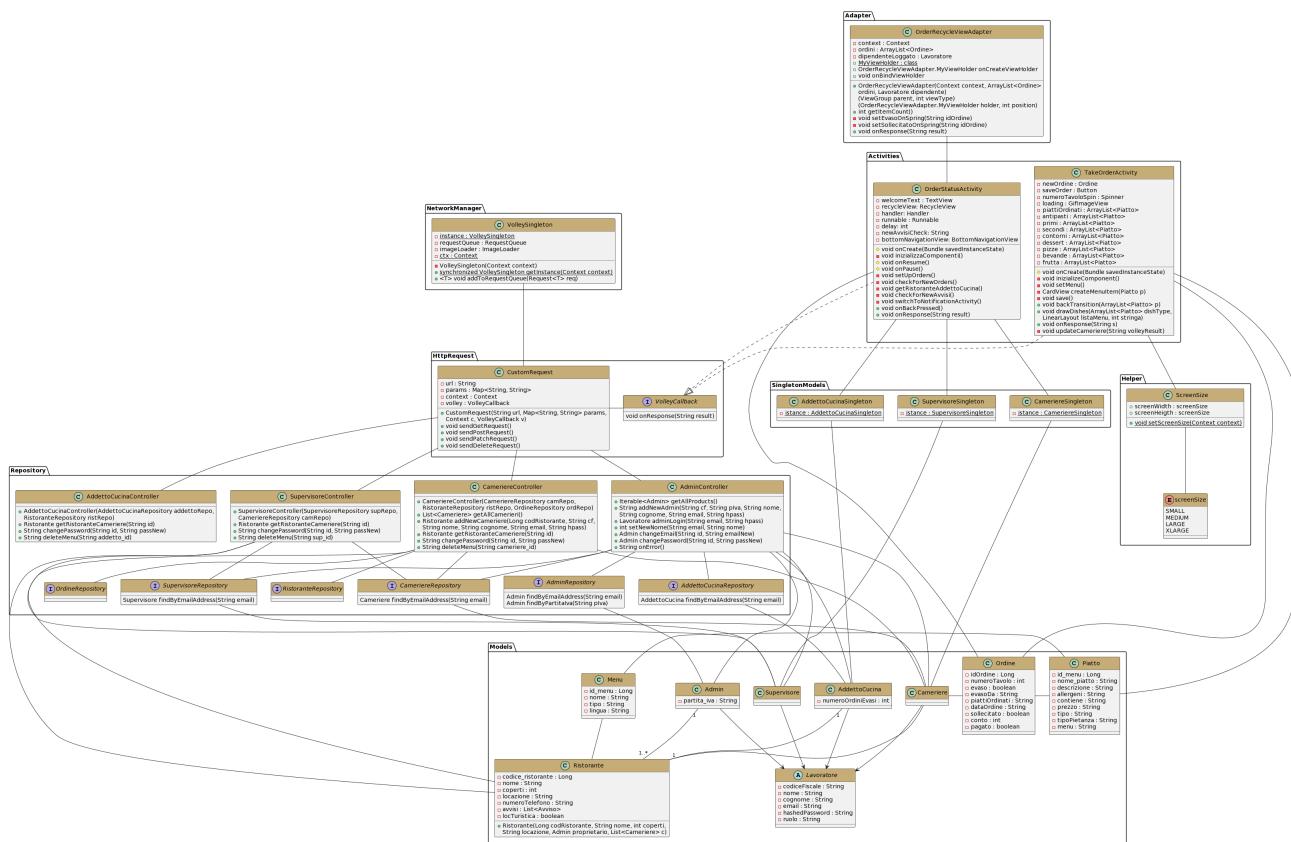
3.3.4 Gestione Avvisi



CD03: Class diagram gestione avvisi

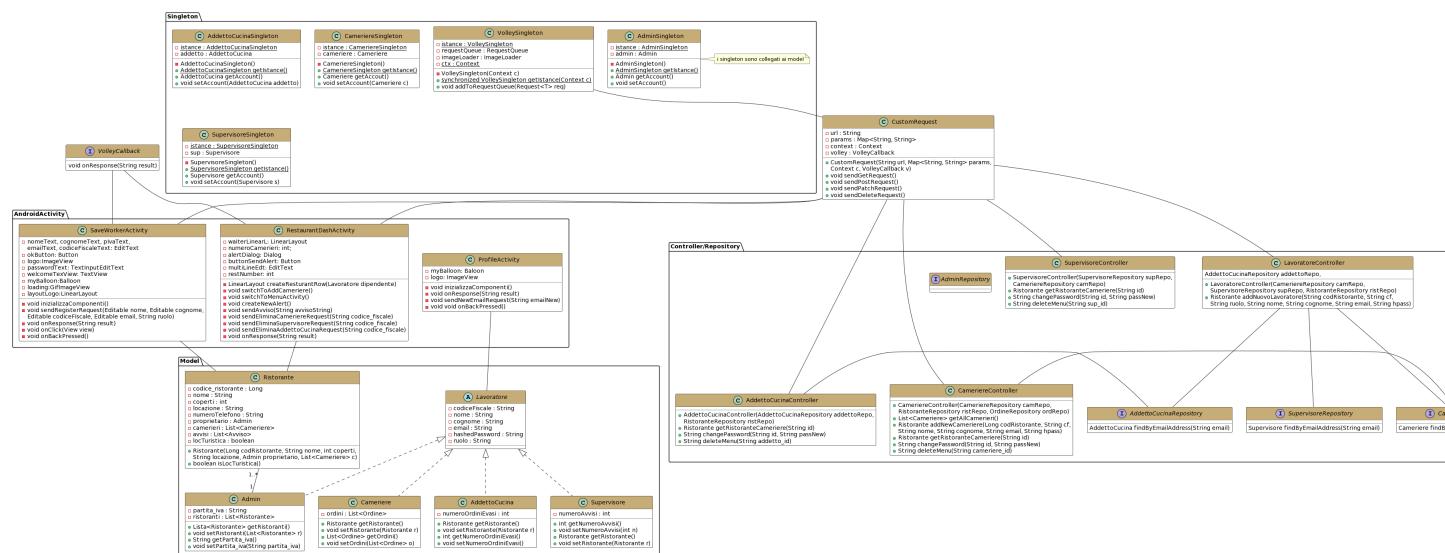


3.3.5 Gestione Ordini



CD04: Class diagram gestione ordini

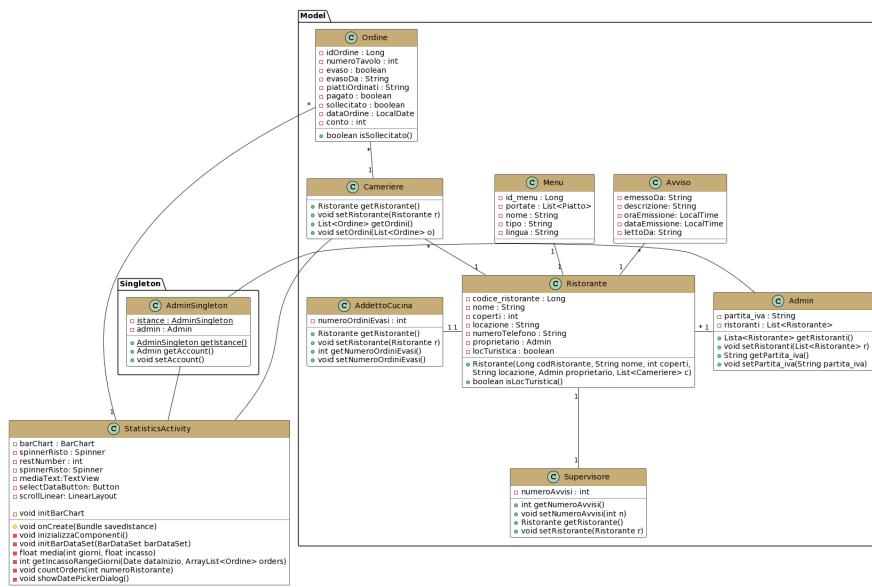
3.3.6 Gestione Dipendenti



CD05: Class diagram gestione Dipendenti



3.3.7 Visualizzazione Statistiche



CD06: Class diagram visualizzazione statistiche



4 Testing JUnit

In questa sezione, presentiamo una delle fasi cruciali all'interno del ciclo di sviluppo del software, ovvero il testing. Il testing può assumere diverse forme e tipologie, ma per la semplicità della presentazione, in questa sede si richiede il testing di soli 4 metodi con due o più parametri, che verrà effettuato mediante l'utilizzo del framework JUnit 5.

4.1 Test funzione check credenziali

Il metodo *checkCredentials* ha il compito di verificare la validità di un indirizzo email e di una password. In particolare, il metodo accetta due stringhe come input: *mail*, che rappresenta l'indirizzo email, e *pswrd*, che rappresenta la password. Il metodo restituisce un oggetto *ArrayList* di interi, che rappresenta una lista di eventuali errori verificatisi durante l'esecuzione del metodo. In altre parole, il metodo controlla se l'indirizzo email e la password rispettano determinati requisiti, e in caso contrario, aggiunge il corrispondente codice di errore alla lista. Infine, il metodo restituisce la lista degli errori, eventualmente vuota se non sono stati riscontrati problemi.

```
19
20 public class CheckCredentialsTest {
21
22     /*
23      CLASSI DI EQUIVALENZA:
24
25      PASSWORD: {VALIDA, VUOTA, TROPPO CORTA, NON RISPETTA LA REGEX}
26          - VALIDA = Almeno 8 caratteri di cui 1 Maiuscola,
27            1 Minuscola, 1 Carattere speciale, 1 numero
28
29      EMAIL: {VALIDA, VUOTA, NON VALIDA}
30          - VALIDA = Deve rispettare la sintassi
31            di un email corretta
32
33 -----
34
35      CODICI DI ERRORE:
36          9 = PASSWORD VUOTA,
37          10 = PASSWORD TROPPO CORTA,
38          11 = PASSWORD NON VALIDA (REGEX)
39          12 = EMAIL VUOTA,
40          13 = EMAIL NON VALIDA
41
42 -----
43
44      STRATEGIE DI TESTING UTILIZZATE:
45          BlackBox secondo il criterio SECT
46
47      CASI DI TESTING INDIVIDUATI:
48          22 in 12 metodi
49
```



```
50----- */
51
52     public ArrayList<Integer> codici_errore = new ArrayList<Integer>();
53
54     @AfterEach
55     public void clearArrayList() {
56         codici_errore.clear();
57     }
58
59     @Test
60     public void testCheckCredentials() {
61         assertEquals(codici_errore, checkCredentials("ser.dimartino@studenti.unina.it", "Password.123"));
62     }
63
64     @Test
65     public void testPasswordVuota() {
66         codici_errore.add(9);
67         assertAll(
68             () -> assertEquals(codici_errore, checkCredentials("fra.cutugno@studenti.unina.it", null)),
69             () -> assertEquals(codici_errore, checkCredentials("lu.starace@studenti.unina.it", ""))
70         );
71     }
72
73     @Test
74     public void testPasswordTroppoCorta() {
75         codici_errore.add(10);
76         assertEquals(codici_errore, checkCredentials("gio.cutolo@studenti.unina.it", "Aa_09."));
77     }
78
79     @Test
80     public void testPasswordNonValida() {
81         codici_errore.add(11);
82         assertAll(
83             () -> assertEquals(codici_errore, checkCredentials("alb.aloisio@studenti.unina.it", "password?123")), // MANCA LA MAIUSCOLA
84             () -> assertEquals(codici_errore, checkCredentials("an.corazza@studenti.unina.it", "PASSWORD#123")), //MANCA LA MINUSCOLA
85             () -> assertEquals(codici_errore, checkCredentials("alb.aloisio@studenti.unina.it", "Password123")), // MANCA IL CARATTERE SPECIALE
86             () -> assertEquals(codici_errore, checkCredentials("an.corazza@studenti.unina.it", "Password_unoduetre")) // MANCA IL NUMERO
87         );
88     }
89 }
```



```
91     @Test
92     public void testEmailVuota() {
93         codici_errore.add(12);
94         assertEquals(
95             () -> assertEquals(codici_errore, checkCredentials(null, "Password.123"))
96             ,
97             () -> assertEquals(codici_errore, checkCredentials("", "Password.123"))
98         );
99     }
100
101    @Test
102    public void testEmailIdNonValida() {
103        codici_errore.add(13);
104        assertEquals(
105            () -> assertEquals(codici_errore, checkCredentials("emailcompletamentebagliata", "Password.123")), // EMAIL
106            // COMPLETAMENTE SBAGLIATA
107            () -> assertEquals(codici_errore, checkCredentials("@gmail.com", "Password.123")), // MANCA LO USERNAME
108            () -> assertEquals(codici_errore, checkCredentials("biagio@.net", "Password.123")), // MANCA IL SECOND LEVEL DOMAIN
109            () -> assertEquals(codici_errore, checkCredentials("matteo@libero.", "Password.123")), // MANCA IL TOP LEVEL DOMAIN
110            () -> assertEquals(codici_errore, checkCredentials("luigivirgilio.it", "Password.123")), //MANCA LA @
111            () -> assertEquals(codici_errore, checkCredentials("MATTEO[BIAGIO]LUIGI@libero.IT", "Password.123")) // LO USERNAME PRESENTA CARATTERI
112            // NON CORRETTI
113        );
114    }
115
116    @Test
117    public void testErroriMultipli_9_12() {
118        codici_errore.add(9);
119        codici_errore.add(12);
120
121        assertEquals(codici_errore, checkCredentials("", ""));
122    }
123
124    @Test
125    public void testErroriMultipli_9_13() {
126        codici_errore.add(9);
127        codici_errore.add(13);
128
129        assertEquals(codici_errore, checkCredentials("@studenti@libero@com", ""));
130    }
131
132    @Test
133    public void testErroriMultipli_10_12() {
```



```
131     codici_errore.add(10);
132     codici_errore.add(12);
133
134     assertEquals(codici_errore, checkCredentials("", "Ab.34"));
135 }
136
137 @Test
138 public void testErroriMultipli_10_13(){
139     codici_errore.add(10);
140     codici_errore.add(13);
141
142     assertEquals(codici_errore, checkCredentials("emailcompletamentebagliata", "Ab
143         .34"));
144 }
145
146 @Test
147 public void testErroriMultipli_11_12(){
148     codici_errore.add(11);
149     codici_errore.add(12);
150
151     assertEquals(codici_errore, checkCredentials("", "Password.Password"));
152 }
153
154 @Test
155 public void testErroriMultipli_11_13(){
156     codici_errore.add(11);
157     codici_errore.add(13);
158
159     assertEquals(codici_errore, checkCredentials("@studenti@libero@com", "@studenti.
160         it"));
161 }
```

4.2 Test funzione che calcola incasso in un range di date.

La funzione "getIncassoRangeGiorni" prende in input una data di inizio e una lista di ordini e calcola l'incasso totale degli ordini che sono stati effettuati tra la data di inizio e la data attuale.

La funzione controlla che la lista degli ordini non sia nulla. In caso contrario, restituisce zero. Successivamente, per ogni ordine nella lista degli ordini, la funzione controlla se la data dell'ordine è compresa tra la data di inizio e la data attuale. Se la data è compresa nel range, l'importo dell'ordine viene aggiunto all'incasso totale. Infine, la funzione restituisce l'incasso totale degli ordini effettuati nel range di tempo specificato.

Dato che la funzione "getIncassoRangeGiorni" fa parte della classe StatisticsActivity e fa uso della classe Ordini, abbiamo creato nel Package "Driver" i seguenti *Mock* per poter eseguire il testing:



```
162 public class StatisticsActivityMock {  
163  
164     public float media(int giorni, float incasso){  
165         float media = 0;  
166         if(giorni < 0) throw new IllegalArgumentException("Il numero di giorni non deve  
167             essere negativo");  
168         if (giorni == 0) throw new ArithmeticException("Il numero di giorni non puo  
169             essere 0");  
170         if (incasso < 0) throw new IllegalArgumentException("l'incasso deve essere  
171             maggiore di 0.");  
172         media = incasso / giorni;  
173         media = Math.round(media * 100) / 100f;  
174  
175         return media;  
176     }  
177  
178     public int getIncassoRangeGiorni(LocalDate dataInizio, ArrayList<OrdineMock> orders){  
179         int incassoTotale = 0;  
180         LocalDate endDate = LocalDate.now();  
181         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");  
182  
183         if (orders == null) return 0;  
184         for (OrdineMock ordine : orders) {  
185             LocalDate orderDate = LocalDate.parse(ordine.getDataOrdine(), formatter);  
186             if (orderDate.isEqual(dataInizio) || orderDate.isAfter(dataInizio) &&  
187                 orderDate.isBefore(endDate)) {  
188                 incassoTotale += ordine.getConto();  
189             }  
190         }  
191     }  
192 }
```

```
192 /*  
193     IN QUESTO CASO LA CLASSE ORDINE MOCK CONTIENE SOLO  
194     GLI ATTRIBUTI DI ORDINE CHE ENTRANO IN GIOCO ALL'INTERNO  
195     DEL METODO getIncassoRangeGiorni  
196 */  
197  
198 public class OrdineMock {  
199  
200     private int conto;  
201     private String dataOrdine;  
202  
203     public OrdineMock(int conto, String dataOrdine) {  
204 }
```



```
205     this.conto = conto;
206     this.dataOrdine = dataOrdine;
207 }
208
209 public int getConto() {
210     return conto;
211 }
212
213 public String getDataOrdine() {
214     return dataOrdine;
215 }
216 }
```

```
218 /*
219  * CLASSI DI EQUIVALENZA:
220
221     DATA_INIZIO: {VALIDA, NULL}
222         - Valida: a sua volta puo' essere
223             1.1) Verosimile e precedente agli ordini
224             1.2) Futura agli ordini
225             1.3) Inverosimilmente precedente agli ordini
226
227     ORDINI: {VALIDI, NULL, LISTA VUOTA, CON DATA SBAGLIATA}
228
229 -----
230
231     STRATEGIE DI TESTING UTILIZZATE:
232
233         BlackBox secondo il criterio WECT
234
235
236     CASI DI TESTING RITENUTI NECESSARI:
237         {VALIDA , VALIDI} : 1 Caso
238         {VALIDA , NULL} : 1 Caso
239         {VALIDA , LISTA VUOTA} : 1 Caso
240         {VALIDA , CON DATA SBAGLIATA} : 4 Casi
241         {DATA FUTURA , VALIDI} : 1 Caso
242         {DATA INVEROSIMILMENTE PRECEDENTE , VALIDI} : 1 Caso
243         {NULL , VALIDI} : 1 Caso
244
245 -----
246
247 public class getIncassoRangeGiorniTest {
248
249     StatisticsActivityMock statisticsActivityMock;
250     ArrayList<OrdineMock> ordiniM;
251     DateTimeFormatter formatter;
252 }
```



```
253     @Before
254     public void setUp() {
255         statisticsActivityMock = new StatisticsActivityMock();
256         ordiniM = new ArrayList<>();
257         formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
258     }
259
260     @AfterEach
261     public void clearArrayList() {
262         ordiniM.clear();
263     }
264
265
266     @Test
267     public void testGetIncassoRangeGiorni() {
268         ordiniM.add(new OrdineMock(3, "2023-05-04"));
269         ordiniM.add(new OrdineMock(105, "2023-05-04"));
270         ordiniM.add(new OrdineMock(72, "2023-02-04"));
271
272         LocalDate dataInizio = LocalDate.parse("2023-02-05", formatter);
273
274         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
275         assertEquals(108, result);
276     }
277
278     @Test
279     public void testZeroOrdini() {
280         LocalDate dataInizio = LocalDate.parse("2023-02-01", formatter);
281
282         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
283         assertEquals(0, result);
284     }
285
286
287     @Test
288     public void testOrdiniNull() {
289         LocalDate dataInizio = LocalDate.parse("2023-02-01", formatter);
290
291         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, null);
292         assertEquals(0, result);
293     }
294
295
296     @Test
297     public void testDataInizioNull() {
298         ordiniM.add(new OrdineMock(3, "2023-05-04"));
299         ordiniM.add(new OrdineMock(105, "2023-05-04"));
300         ordiniM.add(new OrdineMock(72, "2023-02-04"));
301     }
```



```
302     assertsThrows(NullPointerException.class,
303                     () -> statisticsActivityMock.getIncassoRangeGiorni(null, ordiniM));
304     }
305
306     @Test
307     public void testDataInizioFutura() {
308         ordiniM.add(new OrdineMock(3, "2023-05-04"));
309         ordiniM.add(new OrdineMock(105, "2023-05-04"));
310         ordiniM.add(new OrdineMock(72, "2023-02-04"));
311
312         LocalDate dataInizio = LocalDate.parse("2033-02-05", formatter);
313
314         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
315         assertEquals(0, result);
316     }
317
318     @Test
319     public void testDataInizioIrrealistica() {
320         ordiniM.add(new OrdineMock(3, "2023-05-04"));
321         ordiniM.add(new OrdineMock(105, "2023-05-04"));
322         ordiniM.add(new OrdineMock(72, "2023-02-04"));
323
324         LocalDate dataInizio = LocalDate.parse("0133-02-05", formatter);
325
326         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
327         assertEquals(180, result);
328     }
329
330     @Test
331     public void testDataOrdiniInFormatoSbagliato(){
332         ArrayList<OrdineMock> ordiniM_1 = new ArrayList<>();
333         ArrayList<OrdineMock> ordiniM_2 = new ArrayList<>();
334         ArrayList<OrdineMock> ordiniM_3 = new ArrayList<>();
335         ArrayList<OrdineMock> ordiniM_4 = new ArrayList<>();
336
337         ordiniM_1.add(new OrdineMock(3, "2023"));
338         ordiniM_2.add(new OrdineMock(105, "02/05/2023"));
339         ordiniM_3.add(new OrdineMock(72, "due-aprile-2023"));
340         ordiniM_4.add(new OrdineMock(105, "2023-32-31"));
341
342
343         LocalDate dataInizio = LocalDate.parse("2023-02-01", formatter);
344
345         assertAll(
346             () -> assertsThrows(DateTimeParseException.class,
347                         () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
348                                     ordiniM_1)
349             ,
349             () -> assertsThrows(DateTimeParseException.class,
```



```
350             () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
351                         ordiniM_2)
352         ),
353         () -> assertThrows(DateTimeParseException.class,
354                         () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
355                         ordiniM_3)
356         ),
357         () -> assertThrows(DateTimeException.class,
358                         () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
359                         ordiniM_4)
360     )
361
362     );
363     ordiniM_1.clear();
364     ordiniM_2.clear();
365     ordiniM_3.clear();
366     ordiniM_4.clear();
367 }
```

4.3 Test funzione che controlla i campi di un ristorante.

Questo metodo prende in input quattro stringhe che rappresentano il nome di un ristorante, il numero di coperti, l'indirizzo e il numero di telefono. Il metodo restituisce un oggetto ArrayList di interi che rappresenta una lista di eventuali errori verificatisi durante l'esecuzione del metodo.

```
368 public class getRestaurantFieldsErrorsTest {
369 /*
370  * CLASSI DI EQUIVALENZA:
371  *
372  * NOME: {VALIDO, VUOTO, TROPPO CORTO}
373  *
374  * COPERTI: {VALIDO, VUOTO, FUORI-RANGE, NON VALIDO}
375  *   - FUORI RANGE: <5 && > 1000
376  *   - NON VALIDO: Non composto da soli numeri
377  *
378  * INDIRIZZO: {VALIDO, VUOTO, TROPPO CORTO, NON VALIDO}
379  *   - NON VALIDO: Contiene caratteri speciali
380  *
381  * TELEFONO: {VALIDO, VUOTO, NON VALIDO}
382  *
383  * -----
384  *
385  * CODICI DI ERRORE:
386  * 1 = NOME TROPPO CORTO (1 CARATTERE MINIMO)
387  * 2 = NOME MANCANTE
388  * 3 = NUMERO DI COPERTI MANCANTE
```



```
389     4 = NUMERO DI COPERTI FUORI RANGE
390     5 = INDIRIZZO MANCANTE
391     6 = INDIRIZZO TROPPO CORTA (MINIMO 5 CARATTERI)
392     7 = NUMERO DI TELEFONO MANCANTE
393     8 = NUMERO DI TELEFONO NON VALIDO (10 CIFRE NUMERICHE RICHIESTE)
394     9 = NUMERO DI COPERTI ERRATO
395    10 = INDIRIZZO NON VALIDO
396
397 -----
398
399     STRATEGIE DI TESTING UTILIZZATE:
400         BlackBox secondo il criterio WECT
401
402     CASI DI TESTING RITENUTI NECESSARI:
403     {VALIDO, VALIDO, VALIDO, VALIDO} : 1 Caso
404     {TROPPO CORTO, VALIDO, VALIDO, VALIDO} : 1 Caso
405     {VALIDO, FUORI-RANGE, VALIDO, VALIDO} : 2 Casi
406     {VALIDO, NON VALIDO, VALIDO, VALIDO} : 2 Casi
407     {VALIDO, VALIDO, TROPPO CORTO, VALIDO} : 1 Caso
408     {VALIDO, VALIDO, NON VALIDO, VALIDO} : 1 Caso
409     {VALIDO, VALIDO, VALIDO, NON VALIDO} : 1 Caso
410     {NULL, NULL, NULL, NULL} : 1 Caso
411
412 -----
413     public ArrayList<Integer> codici_errore = new ArrayList<Integer>(); *
414
415
416     // L'ARRAYLIST DEVE ESSERE PULITO OGNI VOLTA CHE VIENE CONCLUSO UN CASO DI TEST
417     @AfterEach
418     public void clearArrayList(){
419         codici_errore.clear();
420     }
421
422     @Test
423     public void testgetRestaurantFieldsError() {
424         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "10", "Via Roma 1", "0123456789");
425         assertEquals(codici_errore, actualErrors); //Funziona poiche non ci sono codici
426         di errore: l'ArrayList risulta vuoto
427     }
428
429     @Test
430     public void testNomeCampoTropppoCorto() {
431         codici_errore.add(1);
432         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("a", "10", "Via Roma
433         1", "0123456789");
434         assertEquals(codici_errore, actualErrors);
435     }
```



```
435     @Test
436     public void testCampoCopertiFuoriRange() {
437         codici_errore.add(4);
438         assertAll(
439             () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
440                             Test", "1", "Via Roma 1", "0123456789")),
441             () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
442                             Test", "100000", "Via Roma 1", "0123456789"))
443         );
444     }
445
446     @Test
447     public void testCampoCopertiNonValido() {
448         codici_errore.add(9);
449         assertAll(
450             () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
451                             Test", "dieci", "Via Roma 1", "0123456789")),
452             () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
453                             Test", "10a", "Via Roma 1", "0123456789"))
454         );
455     }
456
457     @Test
458     public void testLocazioneCampoTroppoCorto() {
459         codici_errore.add(6);
460         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
461             10", "Via", "0123456789");
462         assertEquals(codici_errore, actualErrors);
463     }
464
465     @Test
466     public void testLocazioneNonValido() {
467         codici_errore.add(10);
468         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
469             10", "Via_Napoli?", "0123456789");
470         assertEquals(codici_errore, actualErrors);
471     }
472
473     @Test
474     public void testNumeroTelefonoCampoTroppoCorto() {
475         codici_errore.add(8);
476         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
477             10", "Via Roma 1", "12345678");
478         assertEquals(codici_errore, actualErrors);
479     }
480
481     @Test
```



```
477     public void testTuttiICampiVuoti() {  
478         codici_errore.add(2);  
479         codici_errore.add(3);  
480         codici_errore.add(5);  
481         codici_errore.add(7);  
482         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("", "", "", "");  
483         assertEquals(codici_errore, actualErrors);  
484     }  
485 }
```

4.4 Testing delle funzione media (in statistiche).

La funzione media prende in input il numero di giorni e l'incasso totale di un ristorante in quei giorni. Essa calcola la media giornaliera di incasso dividendo l'incasso totale per il numero di giorni e restituisce il valore ottenuto. Inoltre, la funzione effettua alcune verifiche di validità sui parametri di input, come il controllo che il numero di giorni non sia negativo o pari a zero, e che l'incasso sia maggiore di zero. In caso di violazione di queste condizioni, la funzione lancia un'eccezione per segnalare l'errore.

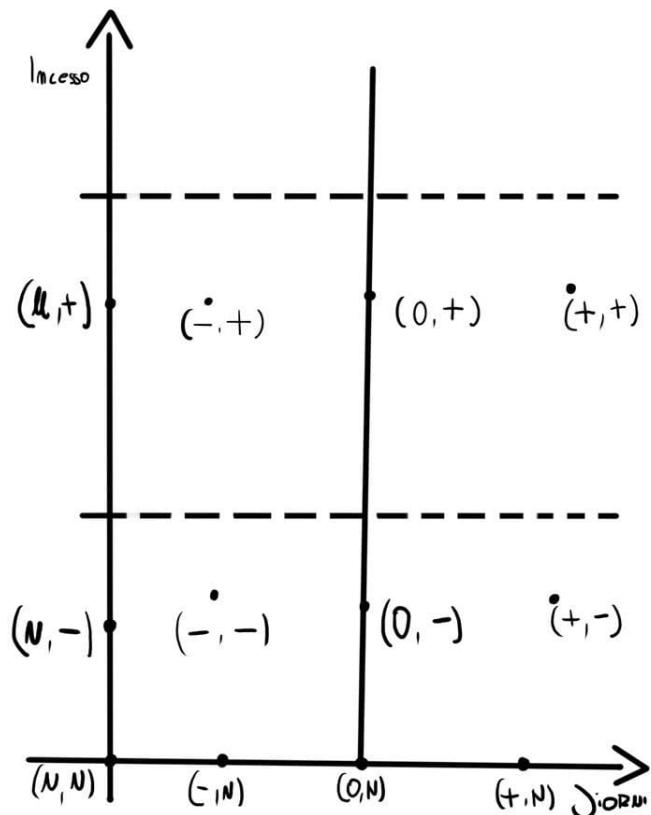


Figura: Prodotto cartesiano CE



```
486 public class mediaTest {  
487     /*  
488      CLASSI DI EQUIVALENZA:  
489  
490      INCASSO: {NULL, NEGATIVO, POSITIVO}  
491      GIORNI: {NULL, NEGATIVO, ZERO, POSITIVO}  
492  
493      -----  
494  
495      STRATEGIE DI TESTING UTILIZZATE:  
496      BlackBox e WhiteBox secondo il criterio SECT  
497  
498      ===== BLACKBOX =====  
499  
500      ----- */  
501  
502     StatisticsActivityMock mock;  
503  
504  
505     @Before  
506     public void setUp(){  
507         mock = new StatisticsActivityMock();  
508     }  
509  
510     @Test  
511     public void testMedia(){  
512         float result = mock.media(17, 50f);  
513         assertEquals(2.94f, result, 0.001f);  
514     }  
515     @Test  
516     public void testGiornoNegativoIncassoPositivo(){  
517         assertThrows(IllegalArgumentException.class,  
518             () -> mock.media(-3, 50.06f)  
519         );  
520     }  
521     @Test  
522     public void testGiornoPositivoIncassoNegativo(){  
523         assertThrows(IllegalArgumentException.class,  
524             () -> mock.media(3, -50.06f)  
525         );  
526     }  
527     @Test  
528     public void testGiornoEIncassoNegativi(){  
529         assertThrows(IllegalArgumentException.class,  
530             () -> mock.media(-3, -50.06f)  
531         );  
532     }  
533     @Test
```



```
534     public void testGiornoZero() {
535         assertAll(
536             () -> assertThrows(ArithmeticException.class,
537                 () -> mock.media(0, 50.06f)
538             ),
539             () -> assertThrows(ArithmeticException.class,
540                 () -> mock.media(0, -50.06f)
541             )
542         );
543     }
```

```
544     /* ===== WHITEKBOX =====
545
546 ----- */
547
548     @Test
549     public void testGiornoNull() {
550         Integer giorno = null;
551         assertAll(
552             () -> assertThrows(NullPointerException.class,
553                 () -> mock.media(giorno, 0.30f)
554             ),
555             () -> assertThrows(NullPointerException.class,
556                 () -> mock.media(giorno, -0.30f)
557             )
558         );
559     }
560     @Test
561     public void testIncassoNull() {
562         Float incasso = null;
563         assertAll(
564             () -> assertThrows(NullPointerException.class,
565                 () -> mock.media(0, incasso)
566             ),
567             () -> assertThrows(NullPointerException.class,
568                 () -> mock.media(3, incasso)
569             ),
570             () -> assertThrows(NullPointerException.class,
571                 () -> mock.media(-3, incasso)
572             )
573         );
574     }
575
576     @Test
577     public void testCampiNull() {
578         Integer giorno = null;
579         Float incasso = null;
580         assertThrows(NullPointerException.class,
```



```
581           () -> mock.media(giorno, incasso)
582       );
583   }
584
585
586 }
```



4.5 Usabilità sul campo

4.5.1 Premessa

Qualunque sia la tecnica utilizzata, i test con gli utenti sono indispensabili. Infatti, le cause delle difficoltà incontrate dagli utenti possono essere moltissime. Analizzare un sistema "a tavolino", come nelle valutazioni euristiche, anche se può permetterci d'individuare numerosi difetti, non è mai sufficiente. I problemi possono essere nascosti e verificarsi soltanto con certi utenti, in relazione alla loro esperienza o formazione. Cose ovvie per chi già conosce il sistema o sistemi analoghi possono rivelarsi difficoltà insormontabili per utenti meno esperti. Un test di usabilità ben condotto mette subito in evidenza queste difficoltà. La necessità del coinvolgimento degli utenti è affermata con chiarezza dalla stessa ISO 13407: "*La valutazione condotta soltanto da esperti, senza il coinvolgimento degli utenti, può essere veloce ed economica, e permettere di identificare i problemi maggiori, ma non basta a garantire il successo di un sistema interattivo. La valutazione basata sul coinvolgimento degli utenti permette di ottenere utili indicazioni in ogni fase della progettazione. Nelle fasi iniziali, gli utenti possono essere coinvolti nella valutazione di scenari d'uso, semplici mock-up cartacei o prototipi parziali. Quando le soluzioni di progetto sono più sviluppate, le valutazioni che coinvolgono l'utente si basano su versioni del sistema progressivamente più complete e concrete. Può anche essere utile una valutazione cooperativa, in cui il valutatore discute con l'utente i problemi rilevati.*"

4.5.2 Presentazione degli utenti

Per svolgere questo tipo di test importantissimi a prodotto finito abbiamo assunto che la nostra app fosse in versione "beta" e distribuito a una cerchia ristretta e selezionata di persone l'applicativo. Per avere un'idea dei test di valutazione dell'usabilità sul campo condotti riportiamo, tramite degli alias, quelle che sono state le "interviste" alle vere persone utilizzatrici. Gli intervistati sono **Luigi, Biagio, Matteo e Massimo**: Luigi, famoso imprenditore, voleva aprire un ristorante 3.0 in una zona turistica del proprio paese, e si è subito messo alla ricerca di un team in grado di sostenerlo nella sua missione.

Matteo, conosciuto per essere il "capo" che chiunque desidera. Sa gestire gruppi di persone, è bravo a comunicare e a segnalare le criticà in ambienti lavorativi.

Biagio, gran lavoratore, ama darsi da fare e sfruttare il massimo dalle tecnologie a disposizione per offrire un servizio sempre di qualità. Anni e anni di esperienza come cameriere in hotel e ristoranti fanno di lui la persona perfetta per far parte del team di Luigi.

Massimo, appena diciottenne, si è affacciato al mondo del lavoro e cerca la sua prima esperienza come addetto alla cucina. Scopre che il ristorante di Luigi cerca giovani in gamba!

4.5.3 Il confronto con gli utenti

Abbiamo dotato i dispositivi Android del personale del nuovissimo ristorante di Luigi della nostra applicazione Ratatuill23, e abbiamo monitorato per due giorni i risultati.

Domanda n.1: Luigi, è stato difficile registrarti e registrare il tuo ristorante nell'app?

Per nulla, anzi, la procedura di registrazione era guidata e un simpatico topolino mi ha guidato tra i vari controlli. Ho potuto abilitare l'opzione "zona turistica", ma a cosa serve?

Domanda n.2: Biagio, com'è stato prendere il tuo primo ordine da cellulare?

Molto seprice devo dire, non ho subito capito come rimuovere gli elementi dall'ordine, ma poi ho letto un'indicazione e ora so farlo! Inoltre improvvisamente il telefono ha suonato, era un avviso da Luigi!



Domanda n.3: Massimo, come ti sei trovato con la nostra app?

Effettivamente l'app è minimale e semplice da usare, in alcuni tratti forse un pochino troppo. In ogni caso, una notifica mi ha avvisato non appena è stato registrato un ordine.

Domanda n.4: Matteo, come ti sei trovato con la nostra app?

Come supervisore della mia sala, ad un certo punto ho avuto necessità di avvisare tutti i camerieri di un problema: grazie alla funzione avviso, ci sono riuscito facilmente.

Domanda n.5: Cosa migliorereste nella nostra applicazione?

Luigi: Avrei effettuato una scelta di colori diversa

Biagio: Forse avrei reso più intuitivo cancellare i piatti da un ordine.

Massimo: Non cambierei nulla, fa quello che deve fare, offrendo tutte le funzioni necessarie a una cucina.

Matteo: Sarebbe da incrementare la velocità di risposta in alcuni casi

Emerge da queste mini interviste un indice di gradimento dell'app di circa il 60/70%, da confrontare e confermare con l'uso nel tempo dell'applicativo.

4.5.4 Valutazione finale

Abbiamo notato che i feedback positivi sono stati maggiori rispetto a quelli negativi e ciò rende il team entusiasta del proprio lavoro. Tra quelli negativi, notiamo che sono tutti risolvibili grazie alla scalabilità delle soluzioni tecnologiche adottate e grazie alla possibilità di garantire aggiornamenti futuri in qualsiasi momento e su qualsiasi fronte del sistema.