

# UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE  
DIPARTIMENTO DI INGEGNERIA ELETTRICA E  
TECNOLOGIE DELL'INFORMAZIONE



CORSO DI LAUREA IN INFORMATICA INSEGNAMENTO DI  
INGEGNERIA DEL SOFTWARE ANNO ACCADEMICO 2022/2023

Specifiche, progettazione,  
implementazione e validazione del  
Sistema Informativo “Ratatouille23”

## Autori

Mario De Luca N86/3911  
Alessandro Bonomo N86/3852

# Indice

<b>I Documento dei Requisiti Software</b>	<b>3</b>
<b>1 Analisi dei requisiti</b>	<b>3</b>
1.1 Modellazione dei casi d'uso richiesti . . . . .	3
1.2 Individuazione del target degli utenti . . . . .	4
1.3 Prototipazione visuale via Mock-up dell'interfaccia utente per tutti i casi d'uso . . . . .	7
1.4 Tabelle di Cockburn . . . . .	9
1.5 Mock-up dei casi d'uso descritti nelle tabelle di Cockburn . . . . .	12
1.6 Valutazione dell'usabilità a priori . . . . .	13
1.7 Glossario . . . . .	16
<b>2 Specifica dei Requisiti</b>	<b>18</b>
2.1 Classi, oggetti e relazioni di analisi . . . . .	18
2.2 Diagrammi di sequenza di analisi . . . . .	22
2.3 Prototipazione funzionale via statechart . . . . .	23
<b>II Documento di Design del Sistema</b>	<b>26</b>
<b>3 Analisi dell'architettura</b>	<b>26</b>
3.1 Architettura 3 Tier . . . . .	26
3.2 Architettura modulare con docker . . . . .	27
3.3 Cloud Hosting . . . . .	29
3.4 Fase di deployment . . . . .	29
3.5 Servizi utilizzati in cloud . . . . .	30
<b>4 Motivazione delle scelte adottate</b>	<b>31</b>
4.1 Motivazione delle tecnologie di sviluppo . . . . .	31
4.2 Documentazione del backend . . . . .	31
4.3 Documentazione del frontend . . . . .	35
<b>5 Diagramma delle classi di design</b>	<b>36</b>
5.1 Le Entities . . . . .	36
5.2 I Boundaries . . . . .	37
5.3 Il Controller . . . . .	41
5.4 DAOs . . . . .	42
<b>6 Diagrammi di sequenza di design</b>	<b>43</b>
6.1 aggiungiRistorante . . . . .	43
6.2 getContiUltime24h . . . . .	44
<b>III Testing e valutazione sul campo dell'usabilità</b>	<b>45</b>
<b>7 Codice xUnit Black Box per Unit Testing</b>	<b>45</b>
7.1 Funzione addElemento . . . . .	45
7.2 Funzione scambiaElementi . . . . .	47
7.3 Strategia di testing Black Box . . . . .	47
7.4 Funzione registraUtente . . . . .	48
7.5 Funzione accediUtente . . . . .	50
<b>8 Valutazione dell'usabilità sul campo</b>	<b>51</b>
8.1 Test di valutazione finale . . . . .	51
8.2 Analisi mediante file di log . . . . .	51
8.3 Integrazione con SDK di Firebase . . . . .	52
8.4 Analisi dei dati ottenuti con Google Analytics . . . . .	53
8.5 File di log . . . . .	54

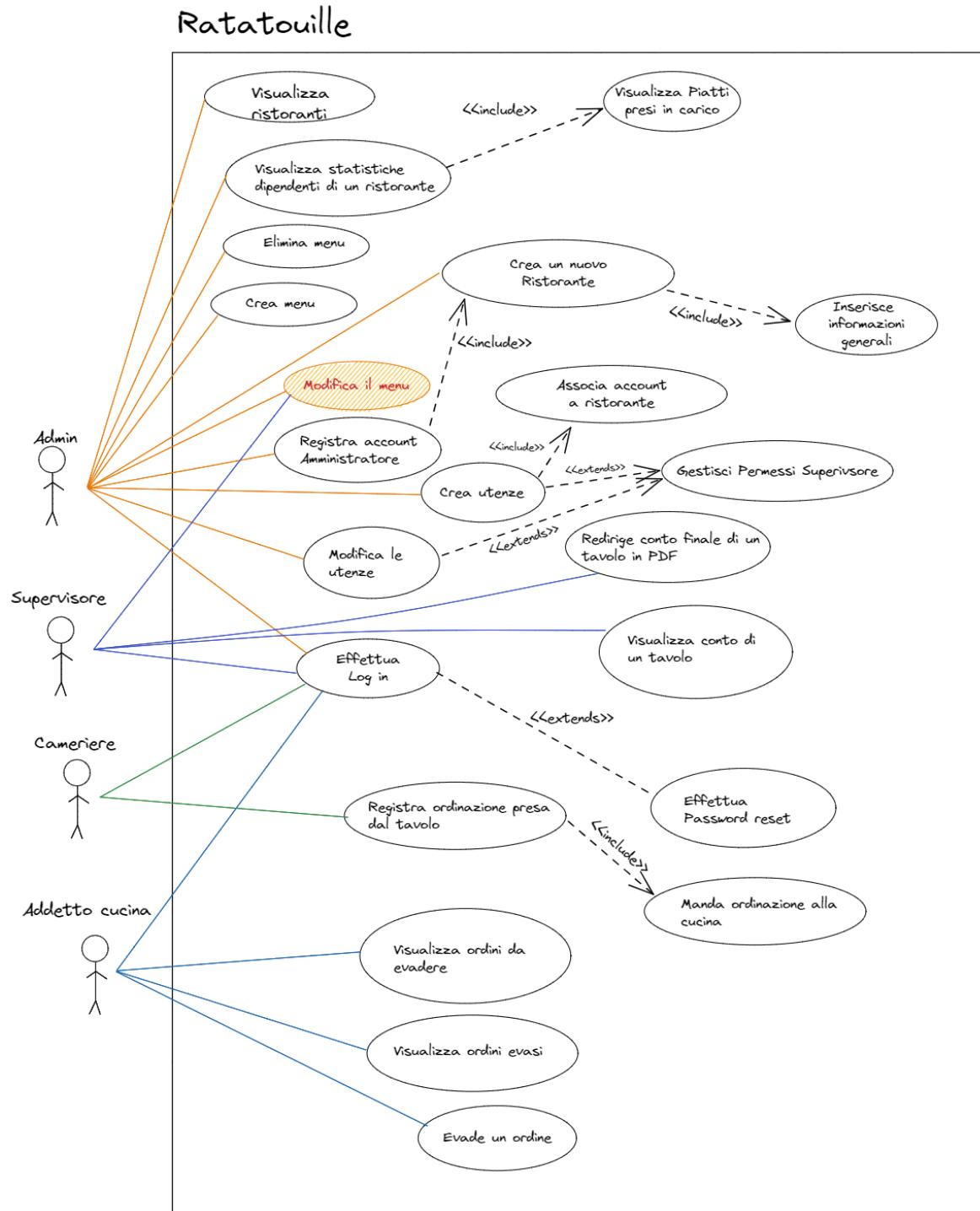
## Sommario

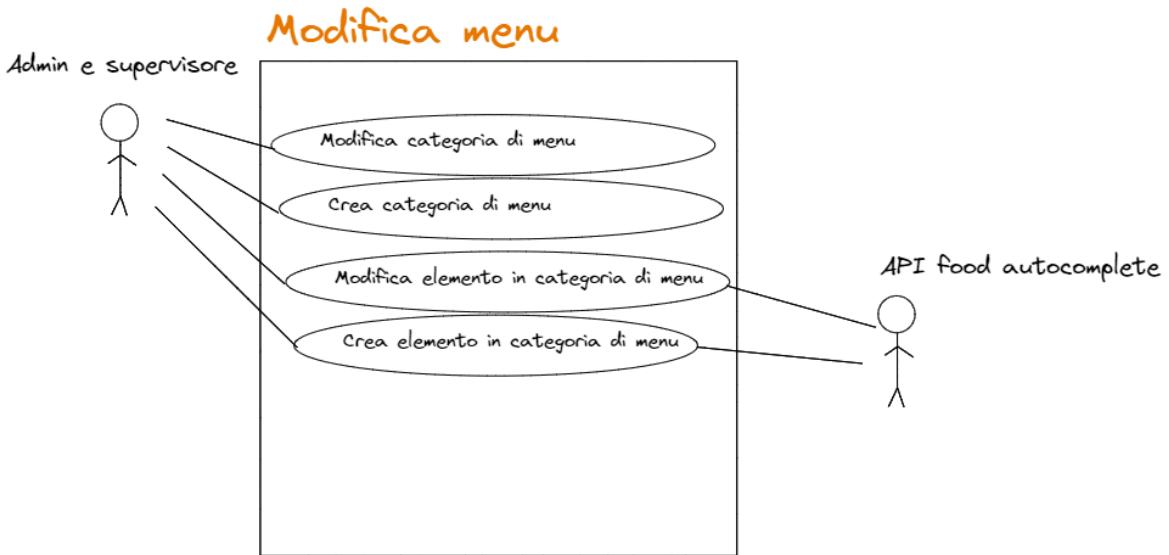
Ratatouille23 è un sistema informativo sviluppato per supportare la gestione e l'operatività di attività di ristorazione. L'obiettivo del team di sviluppo è quello di creare un'applicazione performante e affidabile, dotata di funzionalità intuitive e rapide da utilizzare, al fine di garantire una piacevole esperienza d'uso agli utenti. Tra le principali funzionalità del sistema vi sono la possibilità per gli amministratori di creare utenze per i propri dipendenti, la personalizzazione del menù dell'attività di ristorazione, la registrazione delle ordinazioni e la visualizzazione in tempo reale degli ordini da parte degli addetti alla cucina, la generazione del conto per ogni tavolo e la visualizzazione di statistiche dettagliate sulla produttività del personale addetto alla cucina. Ratatouille23 mira quindi a semplificare e ottimizzare le attività di gestione e operatività all'interno delle attività di ristorazione, fornendo agli utenti un'esperienza d'uso fluida e intuitiva.

# I Documento dei Requisiti Software

## 1 Analisi dei requisiti

### 1.1 Modellazione dei casi d'uso richiesti

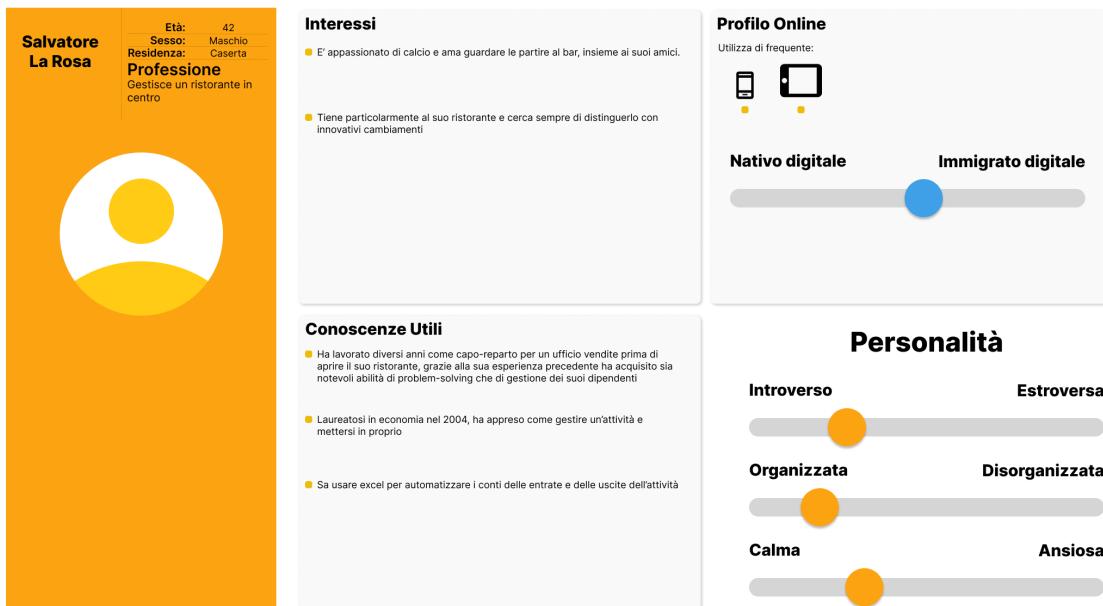




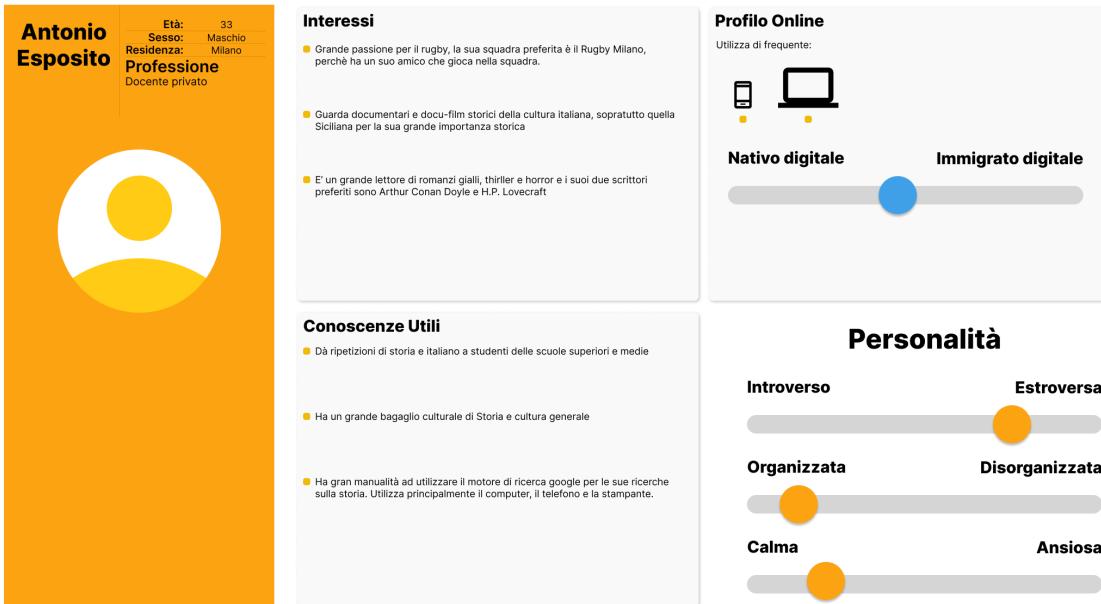
## 1.2 Individuazione del target degli utenti

L'applicazione è stata creata su misura per tutti gli utenti che lavorano nel ambiente della ristorazione. Facilita la gestione del ristorante all'amministratore, aiuta i camerieri a prendere le ordinazioni in maniera più agevole e ordinata. Infine, facilita le interazioni tra gli addetti alla cucina e il personale di sala. Sono state individuate 4 categorie di utenti:

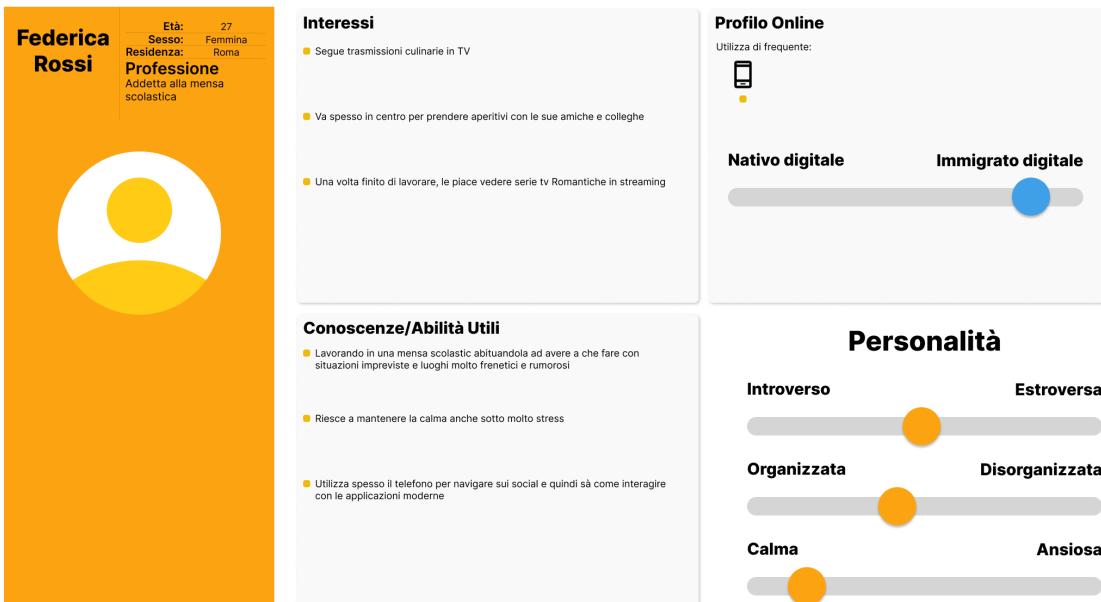
**Amministratore** Colui che gestisce i ristoranti e i suoi dipendenti, può gestire il menù di ogni singolo ristorante e avere un report sulle vendite di ogni locale.



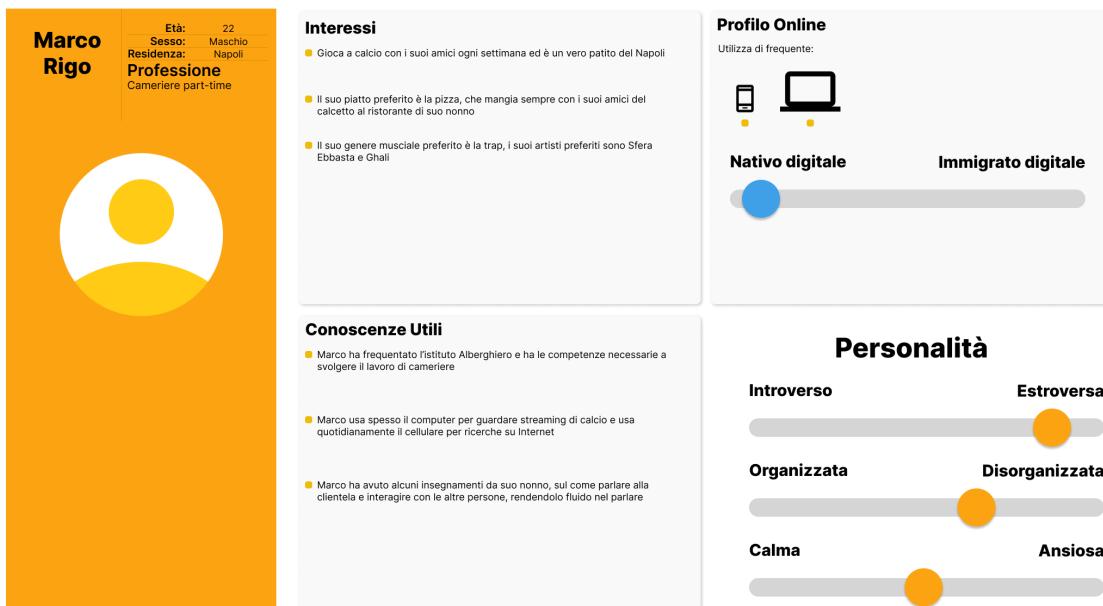
**Supervisore** E' un addetto alla cucina/sala, scelto dall'amministratore per supervisionare i suoi colleghi. Il supervisore può modificare il menu e gestire i conti dei tavoli.



**Addetto alla cucina** E' la persona che si occupa di leggere ed evadere gli ordini ricevuti dal cameriere.



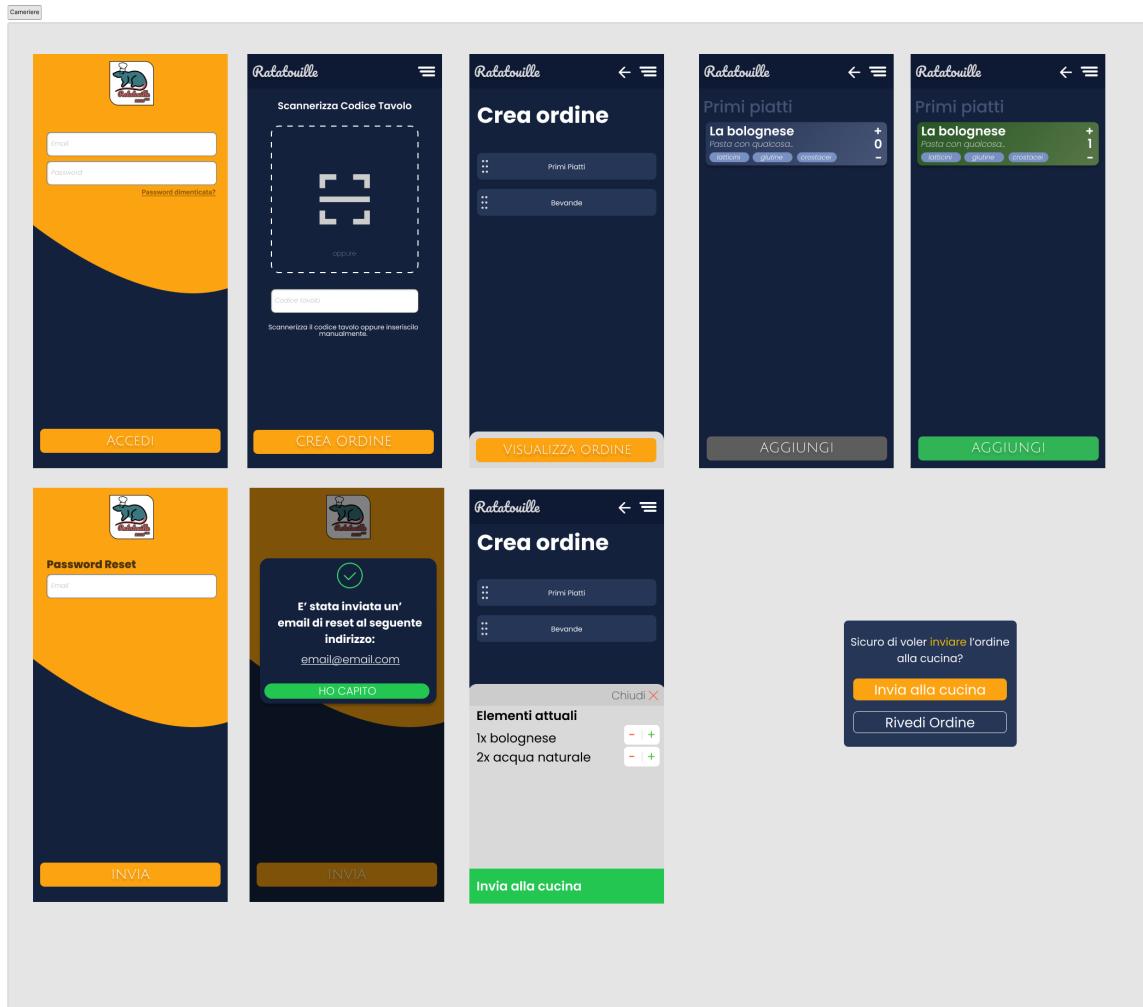
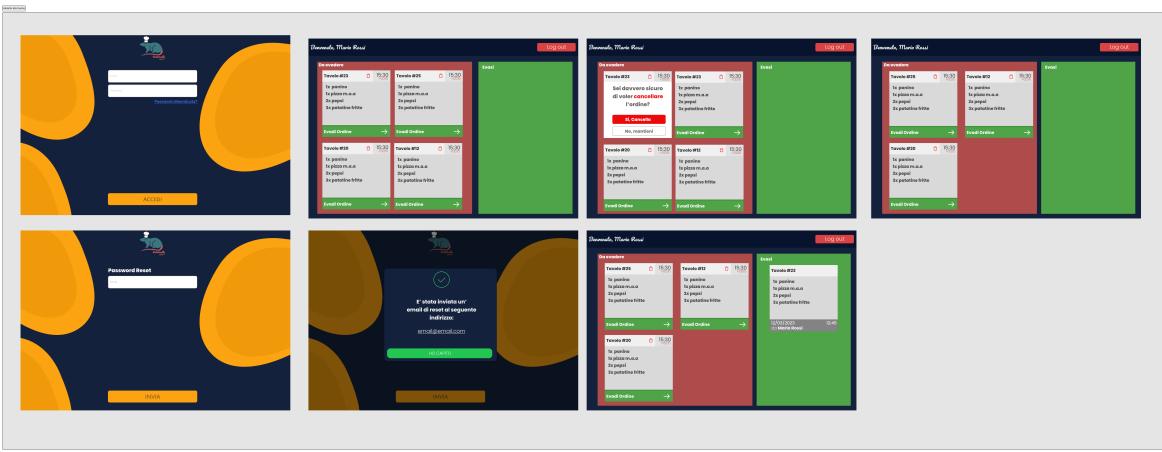
**Addetto alla sala (Cameriere)** E' colui che prende le ordinazioni e le invia alla cucina.



### 1.3 Prototipazione visuale via Mock-up dell'interfaccia utente per tutti i casi d'uso

Si riporta di seguito tutti i mock-up dell'applicazione. I mock up sono stati realizzati con Figma. E' possibile visualizzare i mock up in dettaglio [cliccando qui](#) (Link al progetto Figma in cloud).





## 1.4 Tabelle di Cockburn

Di seguito sono riportate le tabelle di cockburn associate ai relativi casi d'uso:

- Crea nuovo ristorante
- Crea nuova categoria

### 1.4.1 Crea nuovo ristorante

Use Case #1	Crea nuovo ristorante		
Goal in Context	Creazione di un nuovo ristorante con le relative informazioni.		
Preconditions	L'utente deve essere autenticato come amministratore e trovarsi nella schermata Dashboard admin.		
Success End Conditions	Il ristorante viene aggiunto al sistema e viene mostrato nell'elenco dei ristoranti registrati.		
Failed End Conditions	L'utente clicca sull'icona di annullamento e ritorna alla schermata Dashboard admin.		
Primary Actor	Utente amministratore autenticato		
Trigger	L'utente clicca sul pulsante '+' nella schermata Dashboard admin		
Description	Step	User Action	System
	1	Clicca sul pulsante '+' nella schermata Dashboard admin	
	2		Mostra schermata Crea Ristorante
	3	Inserisce nome ristorante	
	4	Inserisce locazione ristorante	
	5	Inserisce numero di telefono ristorante	
	6	Clicca crea	
	7		Torna a Dashboard admin

Extension #1	Step	User Action	System
	1.1	Clicca sulla 'X'	
Extension #2	Step	User Action	System
	6.1	Clicca su 'CREA' senza aver inserito il nome del ristorante	
Extension #3	Step	User Action	System
	6.1	Clicca su 'CREA' senza aver inserito la locazione del ristorante	
Extension #4	Step	User Action	System
	6.1	Clicca su 'CREA' senza aver inserito il numero di telefono del ristorante	
Subvariation #1	Step	User Action	System
	3.1	Inserisce l'url del sito web	
Subvariation #2	Step	User Action	System
	5.1	Seleziona immagine ristorante	
	5.2		Apre il browser del file-system
	5.3	Seleziona l'immagine desiderata e conferma	
	5.4		Mostra preview immagine
	5.4		Vai al punto 6
Notes			

#### 1.4.2 Crea nuova categoria

Use Case #2	Crea nuova categoria		
Goal in Context	Creazione di una nuova categoria del menu.		
Preconditions	L'utente deve essere autenticato come amministratore o supervisore e trovarsi nella schermata Gestione Menu.		
Success End Conditions	La categoria viene aggiunta al sistema e viene mostrata nel menu nella schermata Gestione Menu.		
Failed End Conditions	L'utente clicca sull'icona di annullamento e ritorna alla schermata Gestione Menu.		
Primary Actor	Utente amministratore o supervisore autenticato		
Trigger	L'utente clicca sul pulsante '+' nella schermata Gestione Menu		
Description	Step	User Action	System
	1	Clicca sul pulsante '+' nella schermata Gestione Menu	
	2		Mostra schermata Crea categoria
	3	Inserisce nome della categoria	
	4	Clicca crea	
	5		Torna a schermata Gestione Menu
Extension #1	Step	User Action	System
	1.1	Clicca sulla 'X'	
	1.2		Ritorna alla schermata Gestione Menu
Extension #2	Step	User Action	System
	4.1	Clicca su 'CREA' senza aver inserito il nome della categoria	
	4.2		Mostra messaggio di errore
Notes			

## 1.5 Mock-up dei casi d'uso descritti nelle tabelle di Cockburn

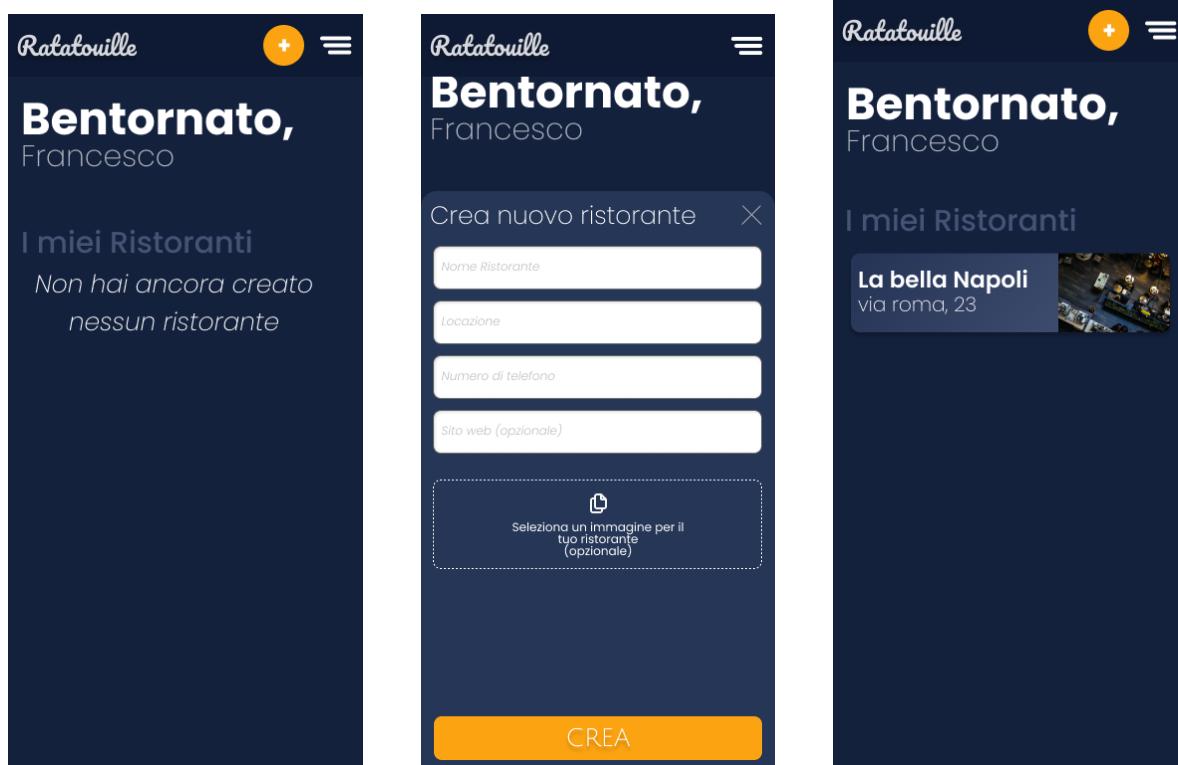




Figura 1: Mock-ups per Crea categoria

## 1.6 Valutazione dell’usabilità a priori

Prima di proseguire con lo sviluppo dell’applicazione, si è deciso di far effettuare ad un gruppo di utenti (tester) una valutazione del prototipo realizzato su Figma.

### 1.6.1 Testing del prototipo con Figma

Sono stati dati ai tester una serie di Task da completare e sono state osservate le loro interazioni e difficoltà nello svolgere i compiti assegnati. E’ stata poi compilata una tabella con gli esiti dei Task dati ai tester. Infine con una semplice formula è stato possibile misurare la facilità di utilizzo dell’applicazione.

Task analizzati:

1. Login
2. Registrazione
3. Crea ristorante
4. Crea ordine e invia alla cucina
5. Completa ed evadi ordine
6. Creazione di un menu con una portata
7. Stampa conto

### 1.6.2 Analisi dei risultati

	1	2	3	4	5	6	7
Tester 1	✓	✓	✓	✗	✓	トラック	✓
Tester 2	✓	✓	✓	トラック	✓	✓	✓
Tester 3	✓	✓	トラック	トラック	✓	✓	✓
Tester 4	✓	✓	✓	トラック	✓	✓	✓

**Leggenda:**

- ✓ Successo +1
- ✗ Fallimento -1
- トラック Successi Parziali +0.5

Ogni tester avrà il suo punteggio associato, che indica la sua facilità di esecuzione di quello specifico task.

$$tasks = 7$$

$$testers = 4$$

$$punteggioMassimo = (testers * tasks) = 28$$

$test_{i,n}$  = esito del test effettuato dal tester "i" nel task "n". Valori possibili: 0.5, 1 , -1

$$punteggioSingoloTester_i = \sum_{n=1}^{tasks} test_{i,n}$$

$$punteggioTotale = \sum_{i=1}^{testers} punteggioSingoloTester_i$$

$$\text{Facilità d' uso } [-1,1] = \text{punteggioTotale}/\text{punteggioMassimo}$$

La facilità d'uso è un numero compreso tra 1 e -1. Se è negativa allora vuol dire che i fallimenti sono maggiori dei successi. Se è positiva, allora i successi sono maggiori dei fallimenti. Se è uguale a 0 allora i successi sono uguali ai fallimenti. Più il punteggio finale si avvicina ad 1, meglio è.

Il punteggio calcolato PRIMA delle correzioni è:

$$\text{punteggioTotale} = 4.5 + 6.5 + 6 + 6.5$$

$$\text{Facilità d' uso} = 23.5/28 = 0.83$$

Il punteggio calcolato DOPO le correzioni è:

$$\text{punteggioTotale} = 6.5 + 6.5 + 7 + 6.5$$

$$\text{Facilità d' uso} = 26.5/28 = 0.94$$

	1	2	3	4	5	6	7
Tester 1	✓	✓	トラック	✓	✓	✓	✓
Tester 2	✓	✓	✓	トラック	✓	✓	✓
Tester 3	✓	✓	✓	✓	✓	✓	✓
Tester 4	✓	✓	✓	✓	✓	トラック	✓

Figura 2: Testing dopo le correzioni

### 1.6.3 Correzioni in base al feedback dei tester

Grazie al feedback dei tester è stato possibile individuare e correggere 2 problematiche:

**Correzione logo** Grazie al suggerimento di uno dei tester è stato deciso di modificare leggermente il logo dell'applicazione per renderlo più affine al contesto culinario.



Figura 3: Correzione del logo. A sinistra la vecchia versione, a destra, la nuova

**Correzione Task N°4 (Crea ordine)** Una difficoltà comune a tutti i tester era riuscire a trovare l'interazione per riuscire a visualizzare e inviare l'ordine alla cucina. E' stato deciso di modificare i Mock up per aumentare l'usabilità dell'applicazione, rendendo più visibile e intuitivo il pulsante per visualizzare l'ordine in corso.

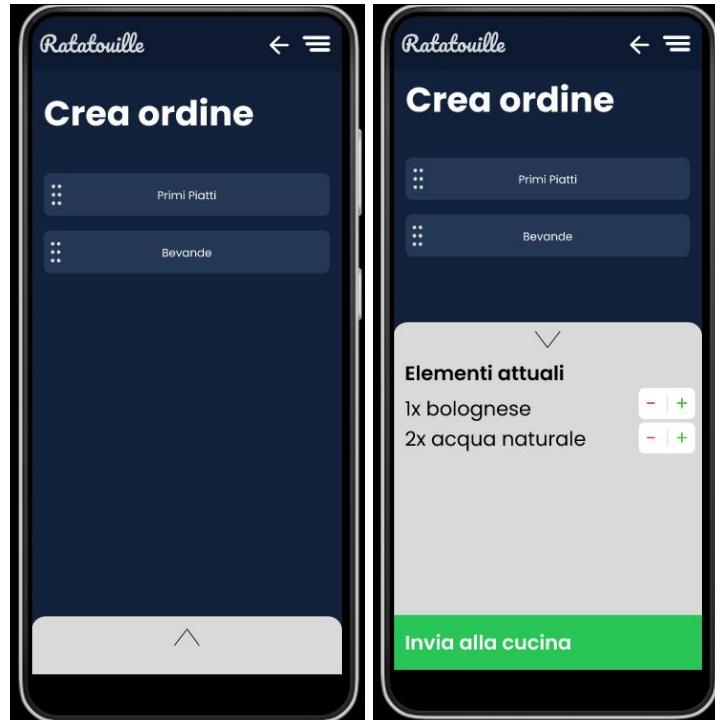


Figura 4: Prima della correzione

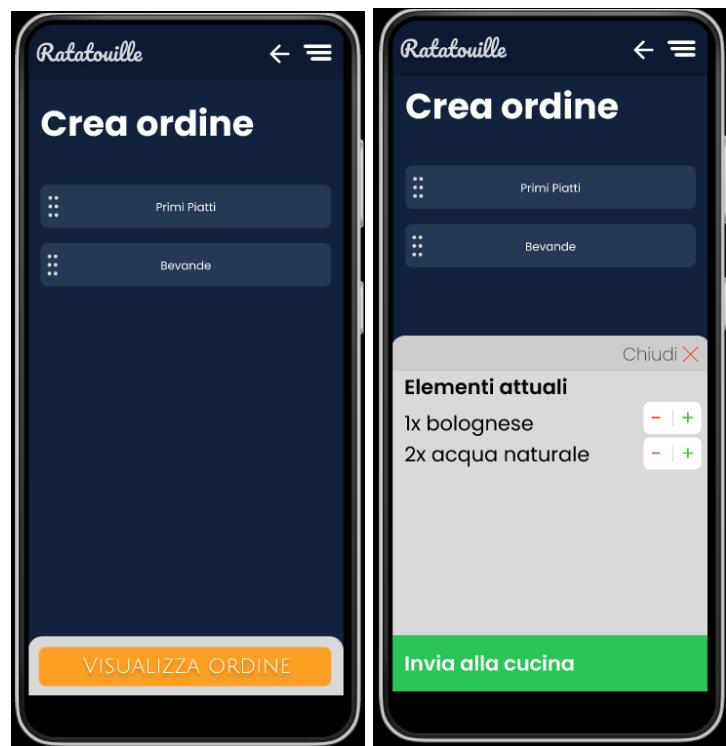


Figura 5: Dopo la correzione

## 1.7 Glossario

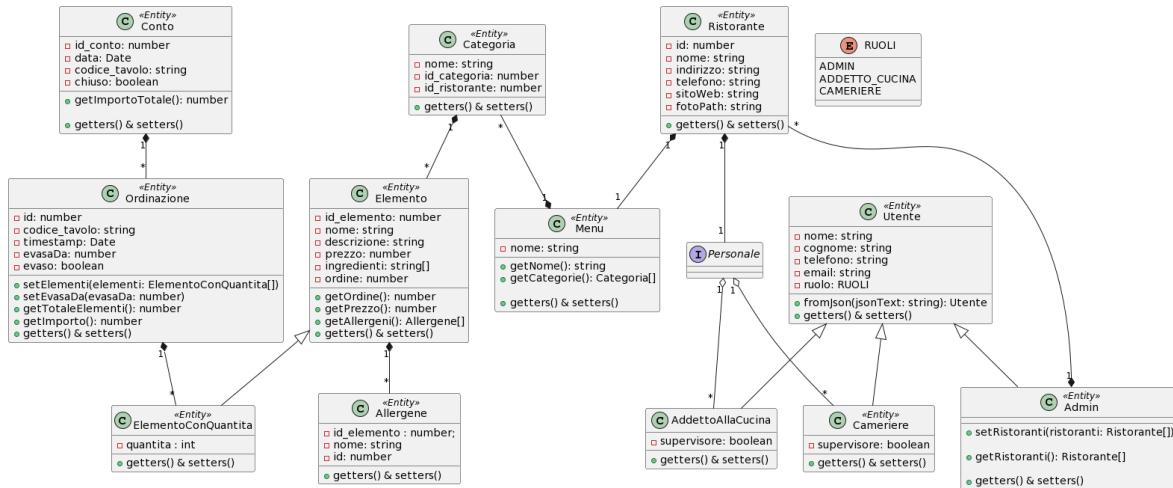
Raccolta dei termini utilizzati all'interno della documentazione:

Termine	Descrizione
Admin / Amministratore	Colui che crea le utenze, visiona le statistiche del personale, crea il menu e amministra i ristoranti
Supervisore	Utente con permessi superiori a quelli dell'utente base ma inferiori all'admin. Può modificare categorie e elementi del menù, stampare il conto e gestire i tavoli e le ordinazioni del locale
Figma	Figma è un software di progettazione grafica basato su cloud, utilizzato principalmente per la creazione di interfacce utente, web design e design di prodotto.
Immigrato digitale	Un immigrato digitale è una persona che ha difficoltà nell'utilizzo della tecnologia digitale e delle piattaforme online.
RESTful API	Un'API RESTful è un'interfaccia di programmazione che utilizza gli standard HTTP per la comunicazione dati.
SSH & SSL	SSH (Secure Shell) è un protocollo per l'accesso remoto sicuro, mentre SSL (Secure Sockets Layer) è un protocollo di crittografia.
Node & Expressjs	Node.js è un runtime JavaScript che consente di eseguire codice lato server. Express è un framework web per Node.js per creare API RESTful.

## 2 Specifica dei Requisiti

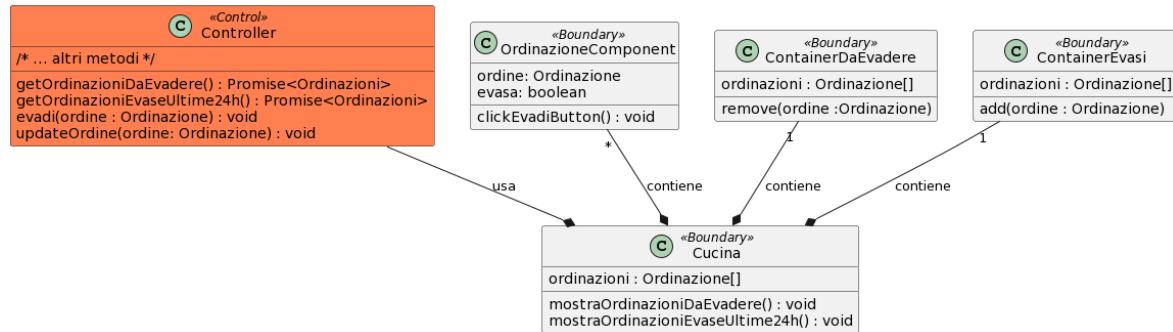
### 2.1 Classi, oggetti e relazioni di analisi

#### 2.1.1 Entities

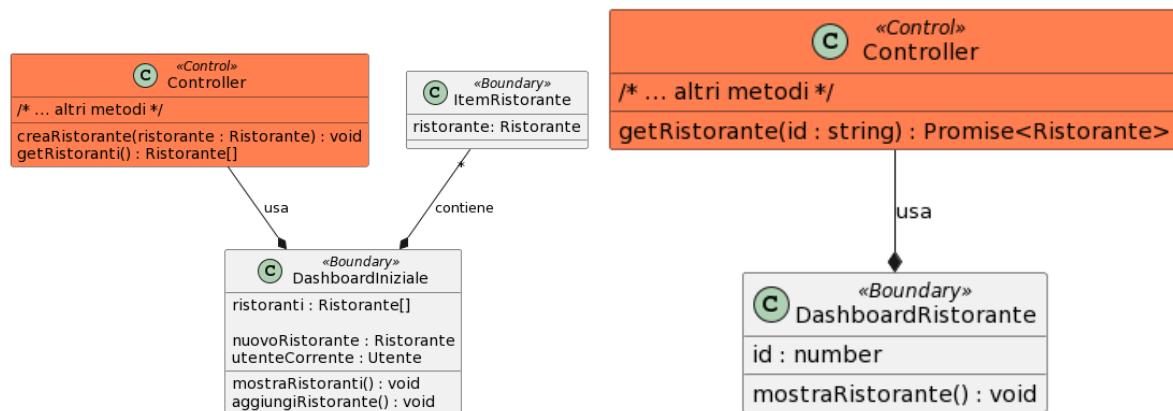


#### 2.1.2 Boundaries

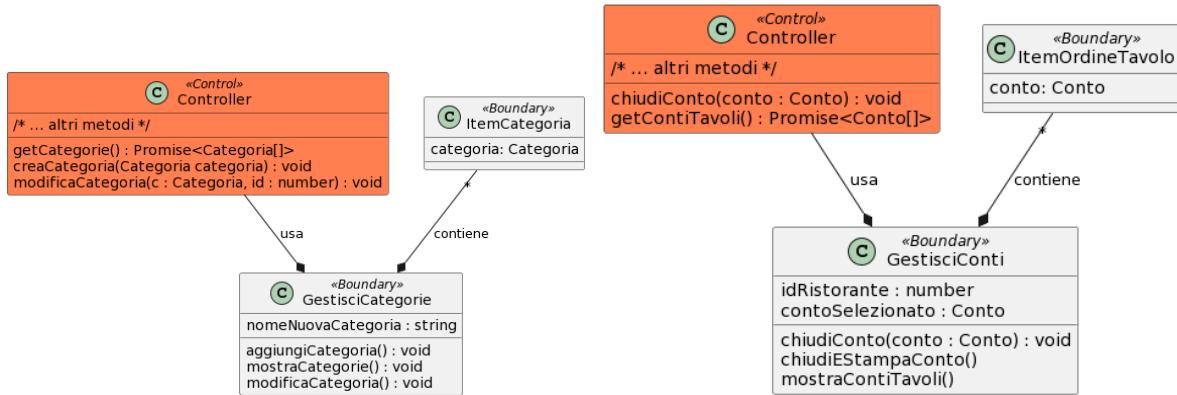
#### 2.1.3 Cucina



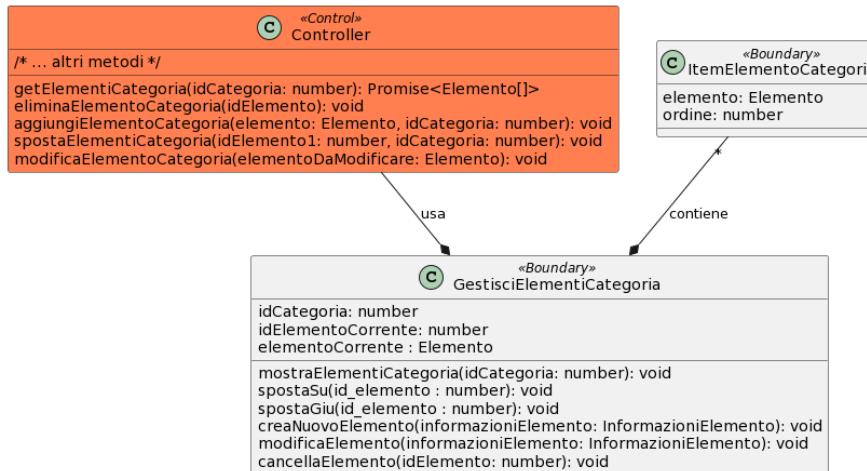
#### 2.1.4 Dashboard iniziale & Dashboard Ristorante



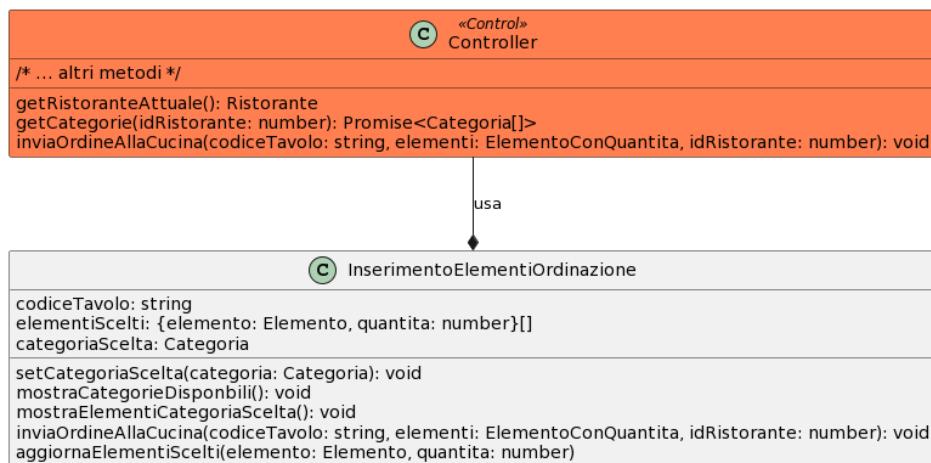
### 2.1.5 Gestisci categorie & Gestisci Conti



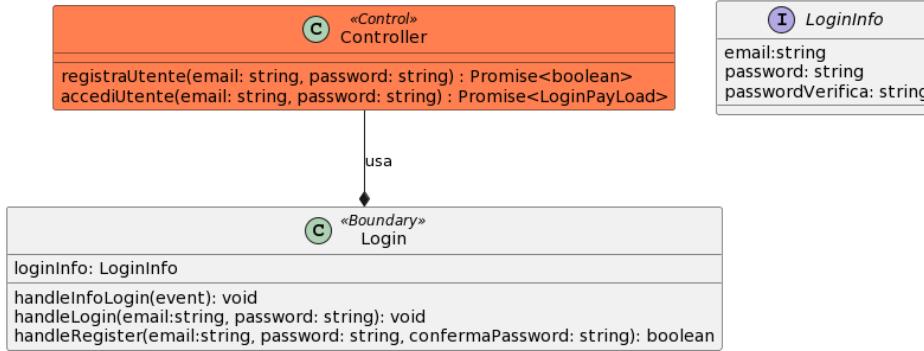
### 2.1.6 Gestisci elementi categoria



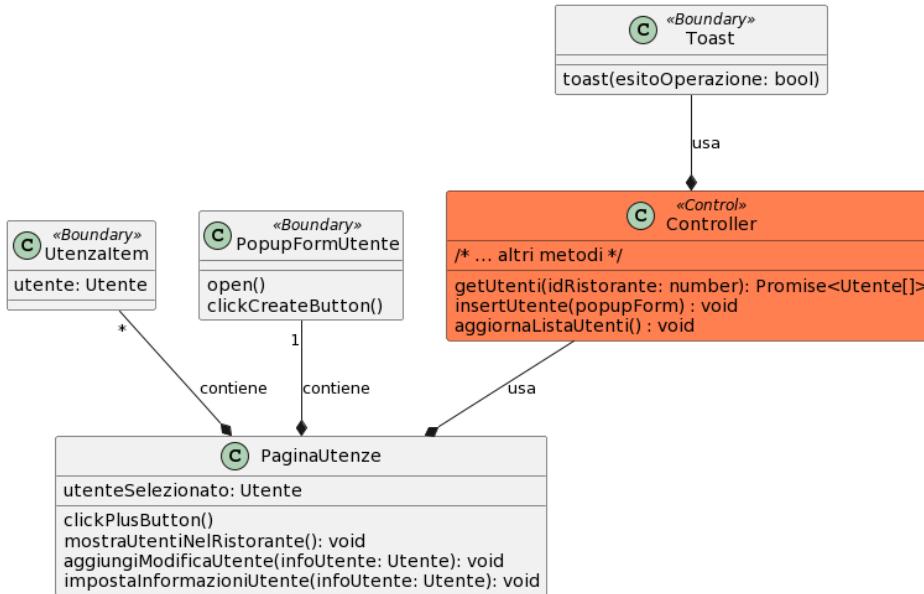
### 2.1.7 Inserimento elementi ordinazione



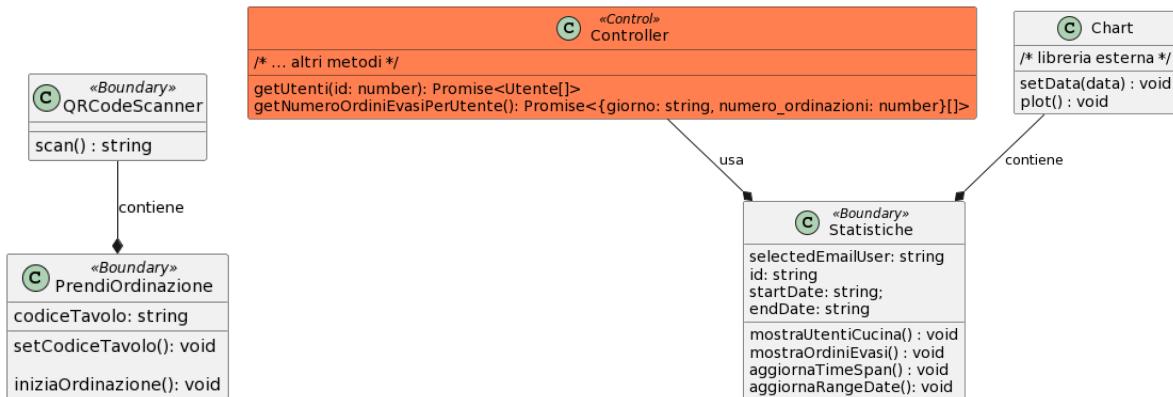
### 2.1.8 Login



### 2.1.9 Pagina Utenze

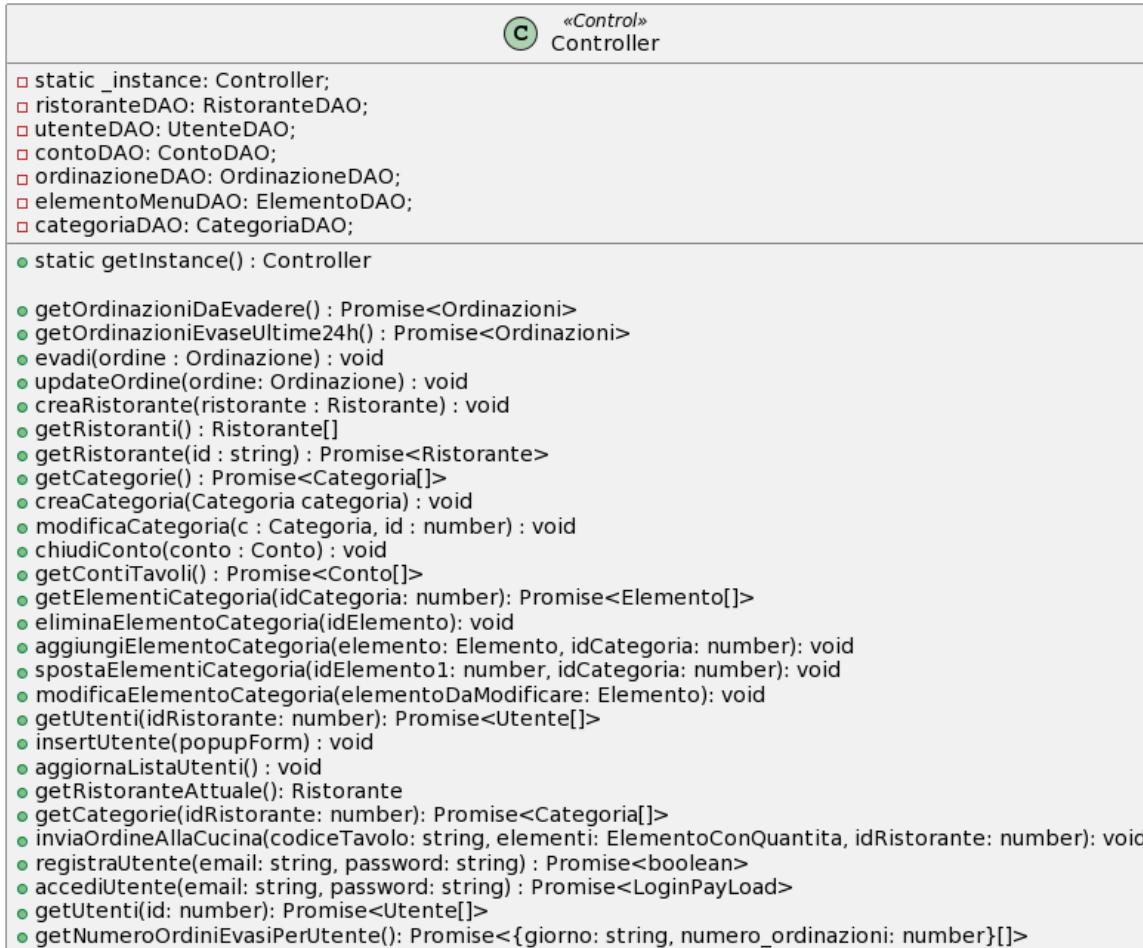


### 2.1.10 Prendi Ordinazione & Statistiche



### 2.1.11 Controller

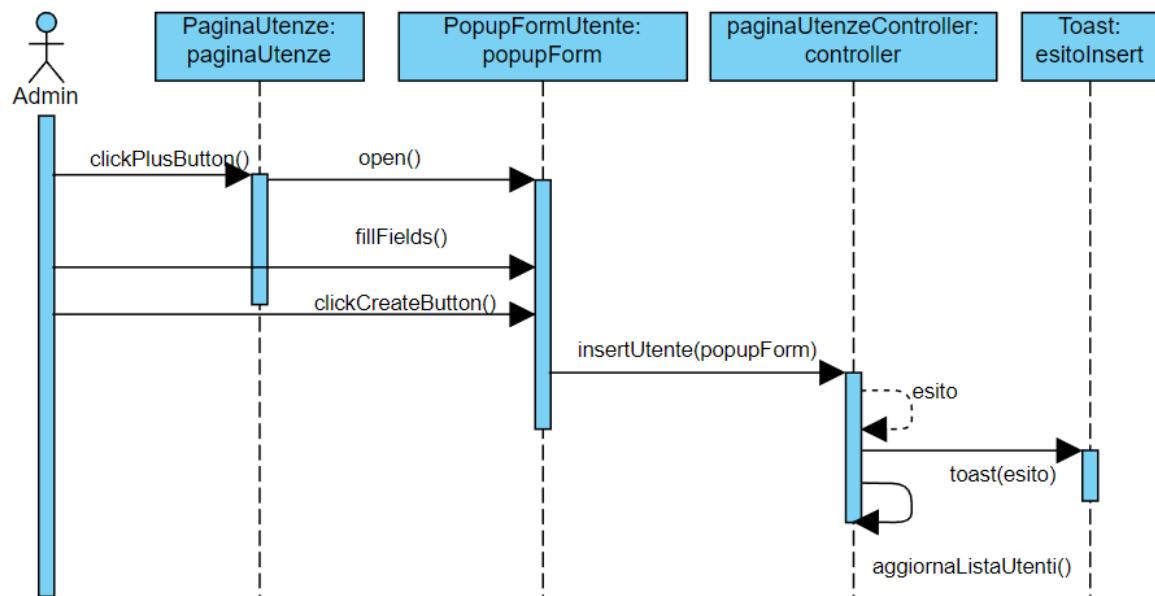
Il controller nell'EBC agisce come un intermediario tra Boundary e Entity, nel funzionamento tipico di un controller EBC (Entity Boundary Control), il Boundary invia una richiesta al controller, che può essere una richiesta di esecuzione di un'operazione, un aggiornamento dei dati o una richiesta di ottenere informazioni. Il controller riceve la richiesta e utilizza le informazioni fornite dal Boundary per prendere decisioni appropriate.



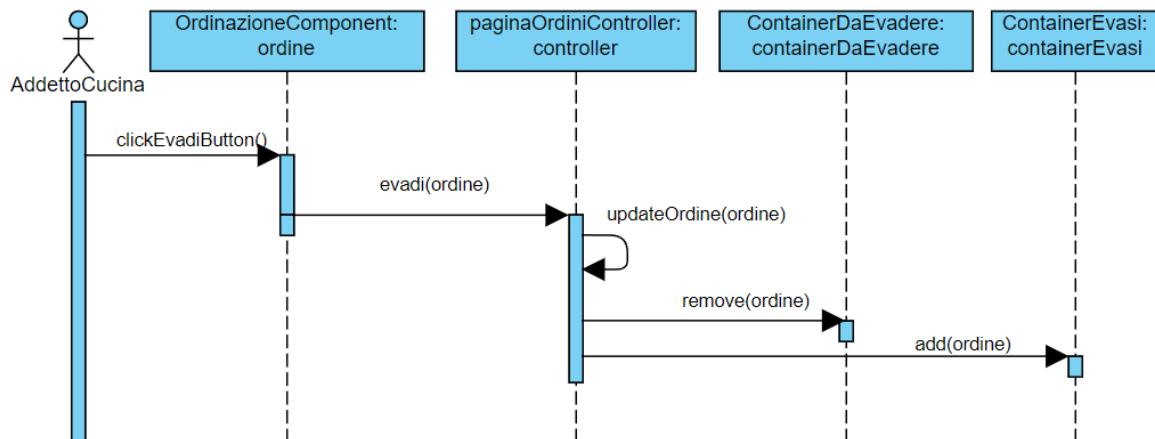
**Utilizzo del design pattern Singleton** Il design pattern Singleton è un pattern creazionale che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a tale istanza. La sua implementazione prevede la definizione di una classe con un costruttore privato e un metodo statico per ottenere l'istanza unica della classe. Il metodo statico controlla se l'istanza è già stata creata, se non lo è, la crea e la restituisce. In questo modo, ogni chiamata al metodo di accesso restituirà sempre la stessa istanza della classe. Il Singleton garantisce l'unicità della classe Controller e permette di accedere ad esso in qualsiasi momento mediante l'invocazione del metodo statico *Controller.getInstance()*.

## 2.2 Diagrammi di sequenza di analisi

### 2.2.1 Crea utenza

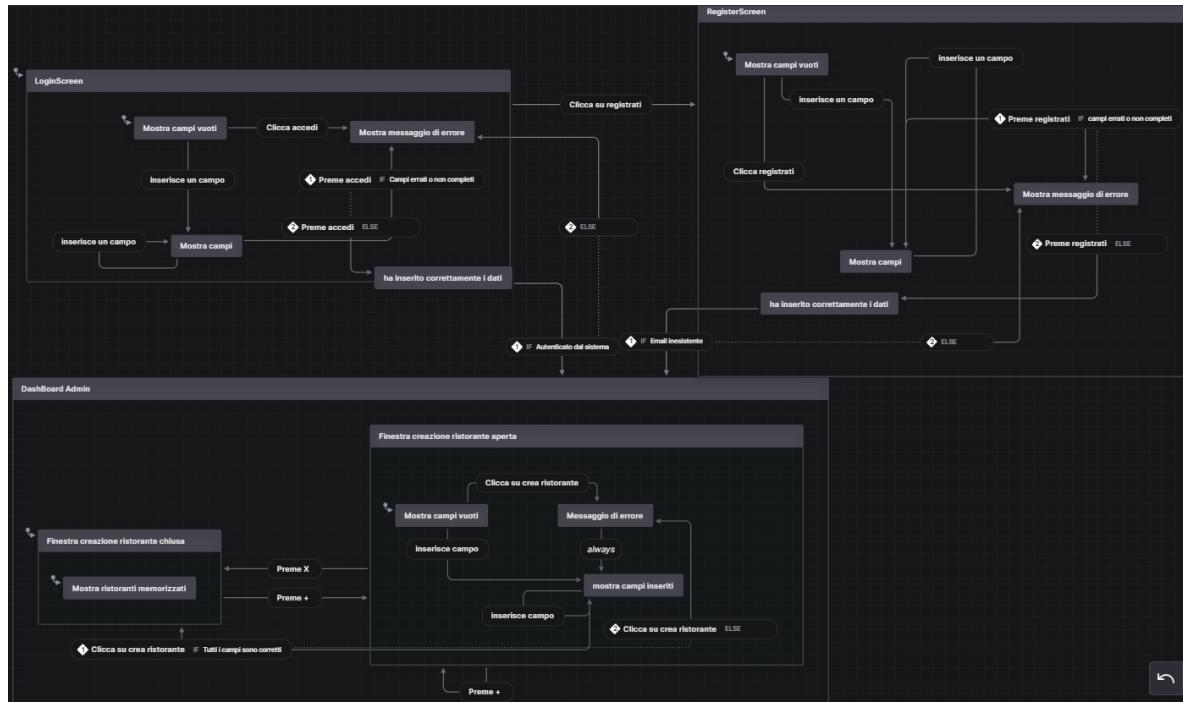


### 2.2.2 Visualizza e evadi ordini

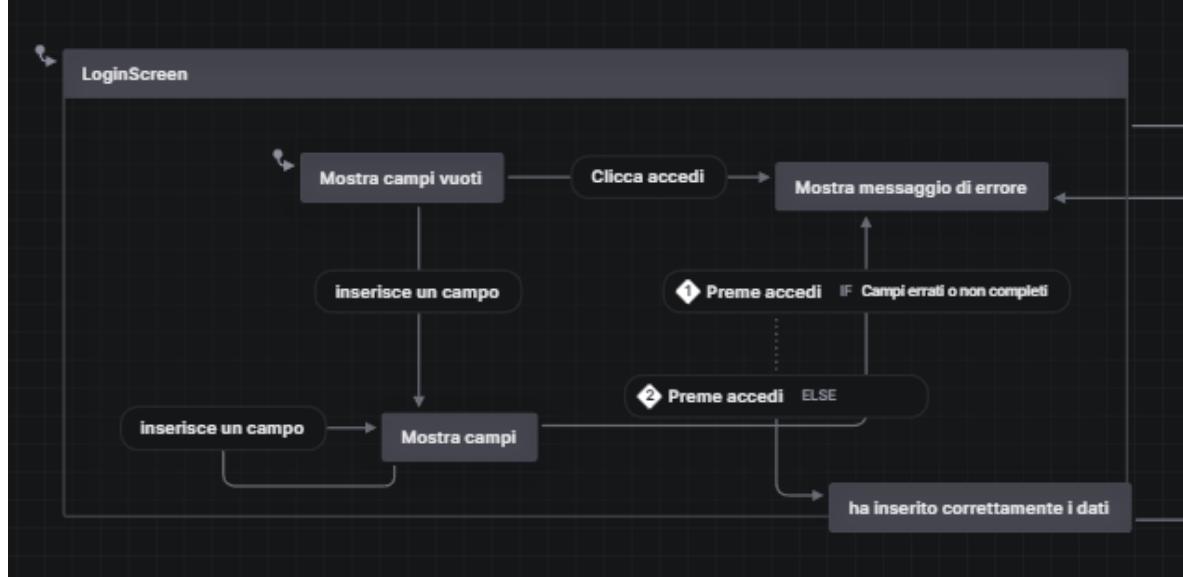


## 2.3 Prototipazione funzionale via statechart

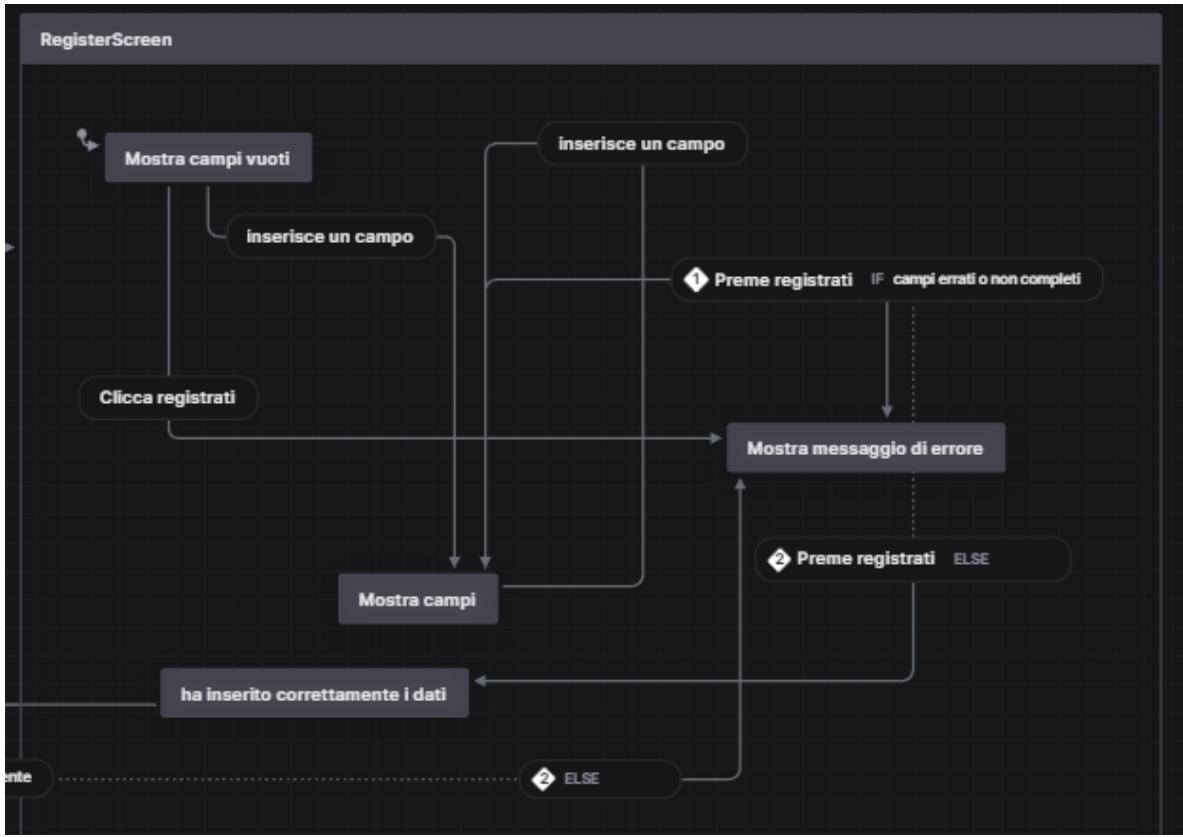
### 2.3.1 Statechart - Crea un Ristorante



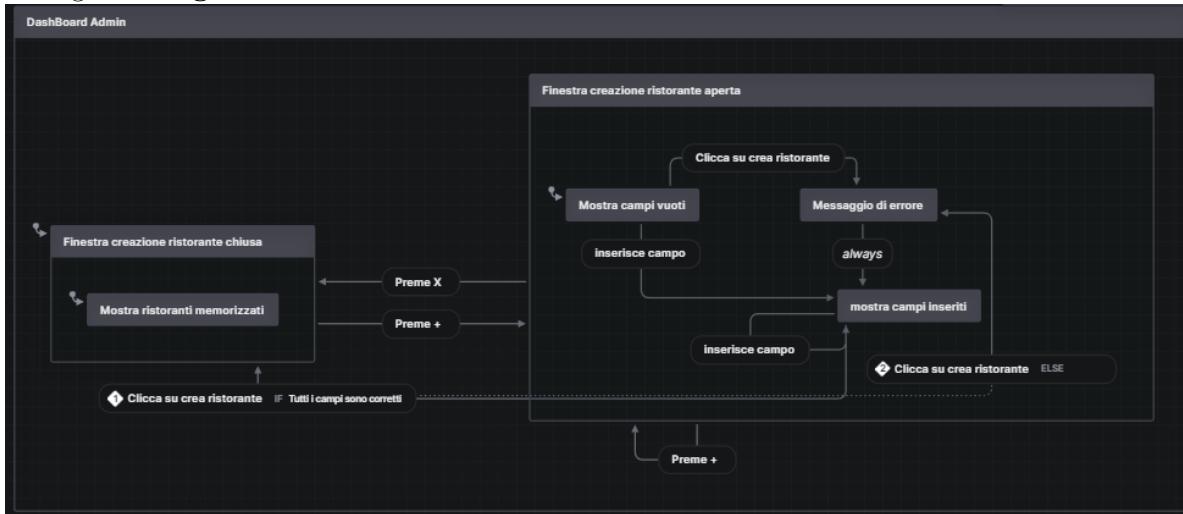
Schema complessivo



Dettaglio su Login

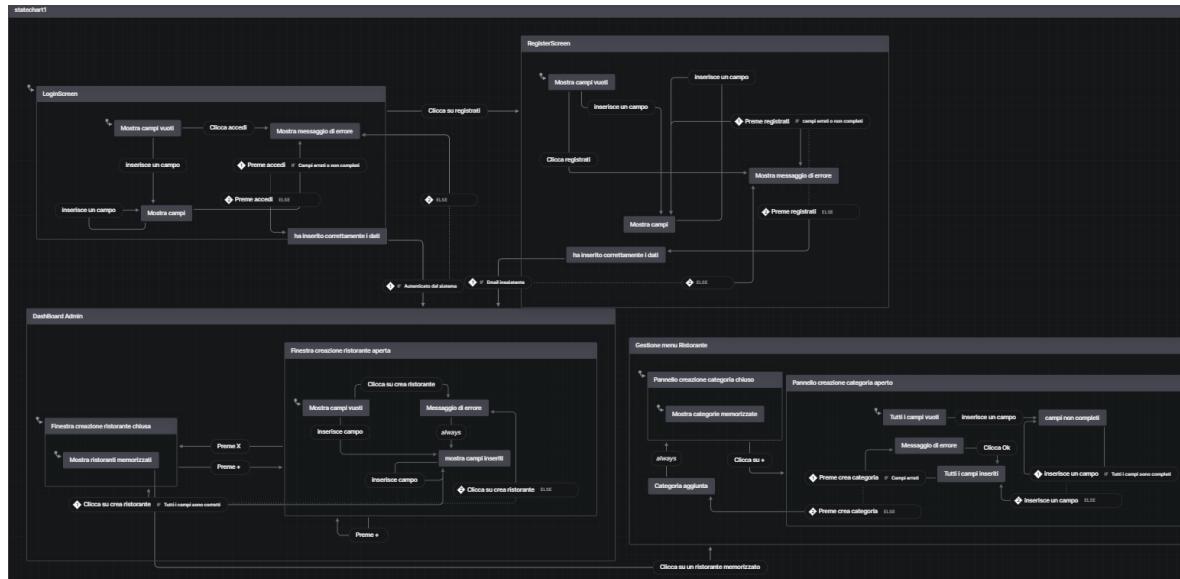


Dettaglio su Register

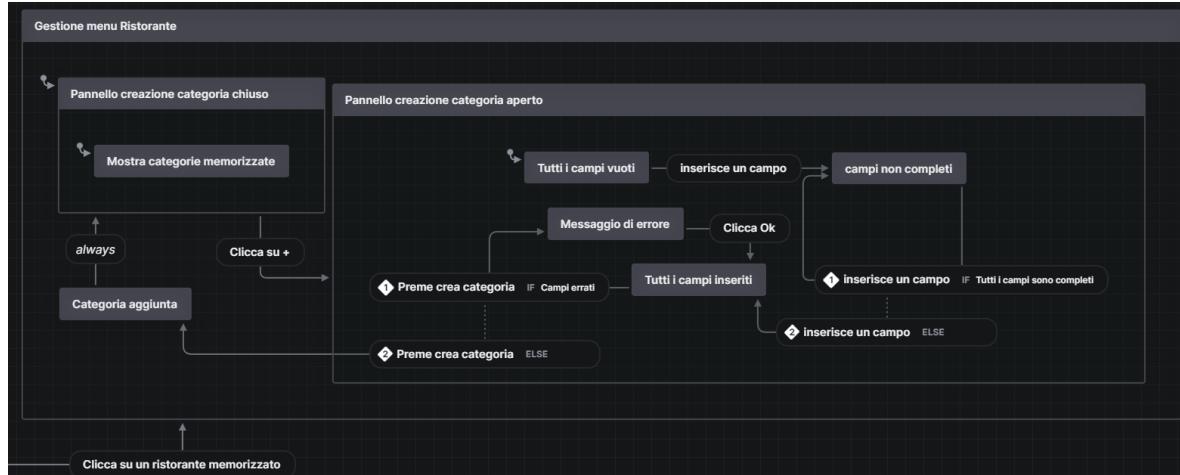


Dettaglio su Dashboard Admin

### 2.3.2 Statechart - Crea una Categoria



Schema complessivo



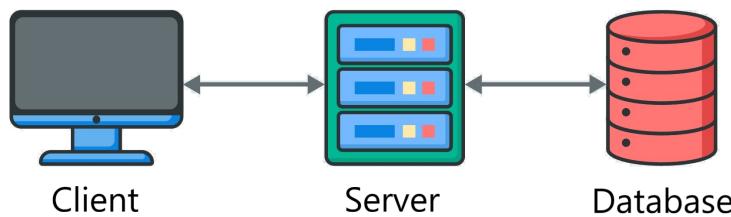
Dettaglio su Gestione menu Ristorante

## II Documento di Design del Sistema

### 3 Analisi dell'architettura

#### 3.1 Architettura 3 Tier

E' stato deciso di utilizzare un'architettura di tipo 3 tier. L'architettura 3 tier (o a 3 livelli) è un'architettura software che prevede la suddivisione dell'applicazione in tre livelli distinti: un livello di presentazione (client frontend), un livello di elaborazione (web server backend) e un livello di persistenza dei dati (information source).



##### 3.1.1 Client - Tier 1

Il frontend, o livello di presentazione, ha lo scopo di creare l'interfaccia utente dell'applicazione, che l'utente finale vedrà e con cui interagirà. Nel caso specifico, il **frontend è stato realizzato utilizzando React**. React consente di creare interfacce utente reattive e performanti, migliorando l'esperienza dell'utente. Inoltre, il frontend dialoga con le API del backend, e con API esterne per l'autocompletamento del testo. Le query alle API sono gestite tramite **React query**, una libreria molto utilizzata allo stato dell'arte che integra meccanismi automatici di caching e semplificazione delle query asincrone ai server. Il frontend non si occupa di elaborare i dati o di effettuare operazioni complesse, ma si limita a presentare i dati e a interagire con gli altri livelli dell'applicazione, senza dover preoccuparsi degli aspetti tecnici sottostanti.

##### 3.1.2 Server - Tier 2

Il livello di backend, o livello di elaborazione, è responsabile dell'elaborazione dei dati e delle logiche di controllo dell'applicazione. Nel caso specifico, è stata realizzata una REST API in **express Node.js**. Una **REST API** (API di tipo Representational State Transfer) è un'interfaccia che permette alle applicazioni di comunicare tra loro, scambiando dati in formato JSON o XML attraverso richieste HTTP. Essa è basata sul concetto di risorse, ovvero oggetti o dati che possono essere richiesti, creati, aggiornati o cancellati. Per garantire la sicurezza delle richieste e delle risposte, è stato utilizzato **JSON Web Token (JWT)**. Un JWT è un token di sicurezza che viene utilizzato per autenticare e autorizzare gli utenti in modo sicuro, senza dover trasmettere le credenziali dell'utente ad ogni richiesta. In definitiva, il livello di backend è responsabile dell'elaborazione dei dati e delle logiche di controllo dell'applicazione, garantendo al contempo la sicurezza e il controllo dell'accesso ai dati e alle funzionalità dell'applicazione stessa.

##### 3.1.3 Database - Tier 3

Il terzo tier dell'architettura a tre livelli è il database PostgreSQL che ha il ruolo di Information source. Il **database PostgreSQL di tipo relazionale** conserva i dati e garantisce la loro integrità grazie ai constraint e trigger definiti nello schema. Il database viene inizializzato con delle istanze predefinite dallo script *init.sql*.

### 3.2 Architettura modulare con docker

Per facilitare il deployment e garantire la modularità e robustezza dell'applicazione è stato deciso di utilizzare **Docker** per dividere l' applicazione in più container con responsabilità separate. Docker permette di isolare i servizi in container, in modo da gestirli in maniera indipendente e controllata.

#### 3.2.1 Schema architettura

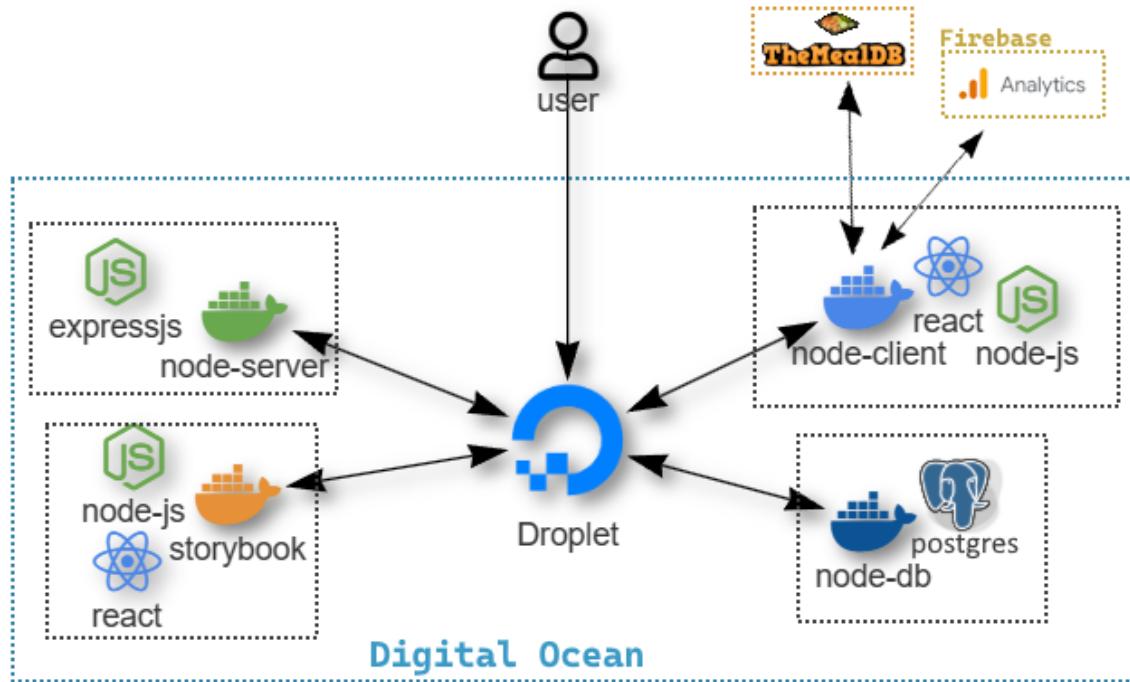


Figura 6: L'utente, con il proprio browser, interagisce con la droplet nel cloud di DigitalOcean. La droplet ospita 4 servizi in dei docker container. Il node-client si interfaccia con google analytics per raccogliere feedback dall'utente e con TheMealDB per l'autocompletamento dei nomi degli alimenti.

#### 3.2.2 Docker containers

Di seguito è possibile osservare tutti e 4 i container attivi. I container si avviano nell'ordine specificato:

1. **node-db**: Database Postgres
2. **node-server**: Backend API express + documentazione API SwaggerUI (Vedi sezione 4.2)
3. **node-client**: Frontend React app
4. **storybook**: documentazione dei componenti del frontend (Vedi sezione 4.3)

Containers <a href="#">Give feedback</a>							
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. <a href="#">Learn more</a>							
<input checked="" type="checkbox"/> Only show running containers		NAME	IMAGE	STATUS	PORT(S)	STARTED	ACTIONS
<input type="checkbox"/>	<input type="checkbox"/>	esame-ingw-22-23	-	Running (4/4)			<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	node-db	306a6169f70f	Running	42069:5432	2 minutes ago	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	node-server	dd975a607af0	Running	3000:3000	2 minutes ago	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	node-client	95313bd9d7ff	Running	3001:3000	2 minutes ago	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	storybook	esame-ingw-22-23-storybook:latest	Running	6006:6006	2 minutes ago	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Nota: storybook è un server per la documentazione del frontend (Vedi storybook nella sezione 4.3)

### 3.2.3 Docker compose file

```
version: '3.9'

services:
  node-frontend:
    container_name: node-client
    restart: on-failure
    build: client/
    ports:
      - "3001:3000"
    depends_on:
      - node-backend
  node-backend:
    container_name: node-server
    restart: on-failure
    build: backend/
    ports:
      - "3000:3000"
    depends_on:
      - node-db

node-db:
  container_name: node-db
  image: postgres
  ports:
    - "42069:5432"
  environment:
    - POSTGRES_USER=docker
    - POSTGRES_PASSWORD=12345
    - POSTGRES_DB=ratatouille
  volumes:
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql
storybook:
  container_name: storybook
  restart: on-failure
  build:
    context: client/
    dockerfile: Dockerfile-storybook
  ports:
    - "6006:6006"
  depends_on:
    - node-frontend
```

### 3.3 Cloud Hosting

**Digital Ocean** è stato scelto come provider di hosting per rendere il software accessibile su Internet. Digital Ocean fornisce un'infrastruttura di hosting affidabile e facile da usare. In particolare, fornisce un indirizzo **IP statico**, garantendo che il sito web sia costantemente accessibile tramite lo stesso indirizzo IP. Per quanto riguarda la configurazione, Docker ha reso la distribuzione dell'applicazione estremamente semplice e portatile, consentendoci con l'esecuzione del *docker-compose* di costruire ed eseguire i container automaticamente. Infine, è stato configurato il nome di **dominio su Aruba: ratatouille.bonomo.cloud** che punta all'IP statico del server Digital Ocean 209.38.197.162 e sono stati aggiunti i certificati SSL per abilitare la comunicazione via **HTTPS**.



### 3.4 Fase di deployment

La fase di deployment consiste nel migrare l'intero insieme di servizi sviluppati in ambiente locale verso il cloud. Per effettuare il deployment dell'applicazione, è stato necessario stabilire una connessione **SSH (Secure Shell)** alla macchina cloud fornita da DigitalOcean. È stato possibile collegarsi alla macchina cloud e avviare le operazioni necessarie per il deployment dell'applicazione. Successivamente, è stato creato un **token Git**, un tipo di token di accesso che consente di autenticarsi a un repository Git senza dover fornire le credenziali di accesso complete. Con il token Git a disposizione, è stato possibile **clonare il repository** dell'applicazione sulla macchina cloud. Una volta ottenuto l'accesso al repository, sono state **installate le dipendenze** necessarie per il corretto funzionamento dell'applicazione. In particolare, Node.js, e **Docker** con Docker Compose, due strumenti ampiamente utilizzati per la creazione e l'esecuzione di container virtualizzati. Infine, con tutte le dipendenze e il repository correttamente configurati, è stato eseguito *docker-compose* compilare tutti i servizi necessari sull'infrastruttura della macchina cloud. Questo processo ha consentito di creare un ambiente di esecuzione completo per l'applicazione, assicurando che tutte le componenti funzionassero correttamente e interagissero tra loro. Al termine dell'esecuzione del *docker-compose*, il deployment dell'applicazione è stato effettuato con successo sulla macchina cloud di DigitalOcean. L'applicazione è ora disponibile per gli utenti e può svolgere tutte le sue funzionalità.

```
❯ ssh root@209.38.197.162
root@209.38.197.162's password:
Welcome to Ubuntu 22.10 (GNU/Linux 5.19.0-43-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Tue Jun  6 14:03:43 UTC 2023

System load:                      0.0087890625
Usage of /:                         38.4% of 24.06GB
Memory usage:                      47%
Swap usage:                        0%
Processes:                          121
Users logged in:                   0
IPv4 address for br-0faffa6b7d46: 172.19.0.1
IPv4 address for docker0:          172.17.0.1
IPv4 address for eth0:             209.38.197.162
IPv4 address for eth0:             10.19.0.5
IPv4 address for eth1:             10.114.0.2
```

(a) Login con SSH

(b) Logo Ubuntu con comando neofetch

```

root@ratatouille-2023-host:~/Esame-INGSW-22-23# ls
DOC_ING_SW_2022_23.pdf  classdiagram.puml  diagramma.png      init.sql          sequence-getContiUltime24h.png
README.md                client           diagrams        package-lock.json   sequenceDiagram.puml
backend                 cockpit.pdf       docker-compose.yml package.json       sequenceDiagram2.puml
'classDiagram IMG.png'  database        immagini      sequenceaggiungiRistorante.png
root@ratatouille-2023-host:~/Esame-INGSW-22-23# docker ps
CONTAINER ID IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
6a1413f8ad6b  esame-ingsw-22-23_storybook "docker-entrypoint.s..." 2 days ago        Up 39 minutes    0.0.0.0:6006->6006/tcp, :::6006->6006/tcp   storybo
ok
c2e7843dfa39  esame-ingsw-22-23_node-frontend "docker-entrypoint.s..." 2 days ago        Up 28 minutes    0.0.0.0:3001->3000/tcp, :::3001->3000/tcp   node-cl
ient
796cb18a8327  esame-ingsw-22-23_node-backend "docker-entrypoint.s..." 2 days ago        Up 30 minutes    0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   node-se
rver
18bf2e00dcfb  postgres         "docker-entrypoint.s..." 2 days ago        Up 28 minutes    0.0.0.0:42069->5432/tcp, :::42069->5432/tcp   node-db
root@ratatouille-2023-host:~/Esame-INGSW-22-23#

```

Figura 8: Containers running in cloud

### 3.4.1 Come l'utente può raggiungere il server

L'utente finale può raggiungere i servizi via dominio *ratatouille.bonomo.cloud* oppure via IP statico:

1. Sito web Ratatouille: <http://209.38.197.162:3001/>
2. Documentazione API : <http://209.38.197.162:3000/doc>
3. Documentazione frontend: <http://209.38.197.162:6006/>

### 3.4.2 Pannello di controllo in cloud

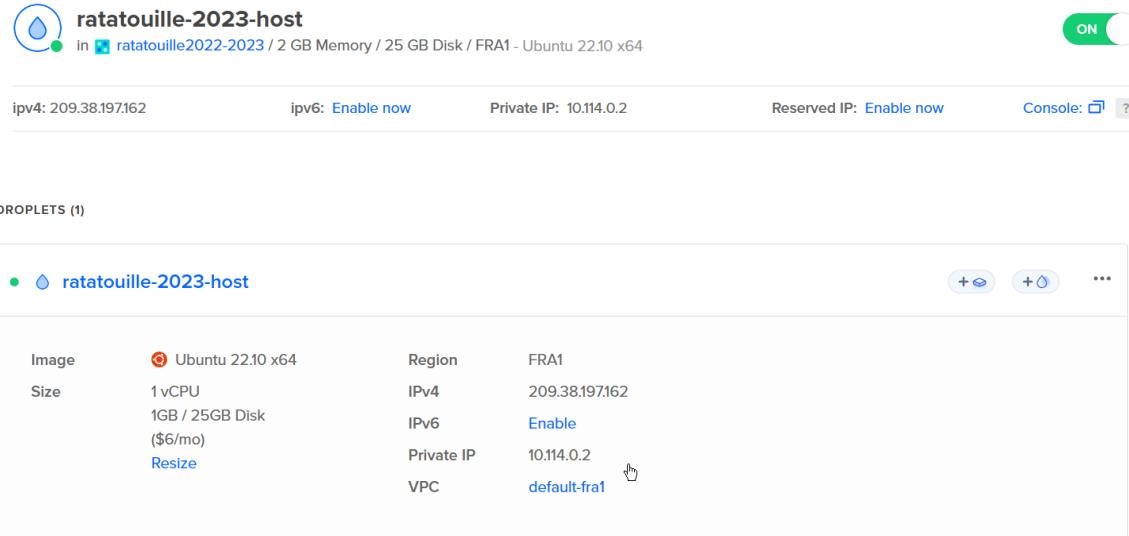


Figura 9: Droplet di DigitalOcean

## 3.5 Servizi utilizzati in cloud

1. **Firebase con Google Analytics**: funzionalità di raccolta dati sugli utenti.
2. **DigitalOcean**: hosting dell'applicativo
3. **TheMealDB**: API food autocomplete

## 4 Motivazione delle scelte adottate

L'**architettura a tre livelli** è stata scelta per organizzare il progetto in modo più strutturato e versatile. In questo modo, ogni livello ha un compito specifico e ben definito, e le interazioni tra di essi sono chiare e gestibili. Inoltre, questo tipo di architettura consente di separare la logica di presentazione dall'elaborazione dei dati, semplificando lo sviluppo e la manutenzione dell'applicazione. Per quanto riguarda la scelta del **cloud Digital Ocean**, è stata fatta considerando diversi fattori come il numero di accessi al mese, il costo, la banda e lo spazio a disposizione. Il cloud offre molte soluzioni di hosting scalabili e personalizzabili, che possono adattarsi alle esigenze dell'applicazione. L'utilizzo di **Docker** è stato scelto per ottenere un'applicazione robusta e facilmente replicabile in diversi ambienti. Docker permette di isolare i servizi in container, in modo da gestirli in maniera indipendente e controllata. Inoltre, Docker garantisce che i servizi vengano avviati nell'ordine corretto, semplificando la configurazione dell'applicazione. Infine, l'uso di Docker consente di avere il controllo completo delle porte esposte, che possono essere gestite in modo flessibile e personalizzato a seconda delle esigenze dell'applicazione.

### 4.1 Motivazione delle tecnologie di sviluppo

#### 4.1.1 React

L'uso di **React per lo sviluppo dell'interfaccia utente** dell'applicazione è stato motivato dalla sua natura dichiarativa e dalla sua architettura basata su componenti riutilizzabili. Grazie a React, è stato possibile suddividere l'interfaccia in parti modulari e indipendenti, semplificando lo sviluppo e la manutenzione del codice. Inoltre, la sua ampia adozione e la vasta comunità di sviluppatori rendono React una scelta affidabile per la creazione di interfacce utente complesse.

#### 4.1.2 React libraries

Per la gestione dello stile dell'applicazione, si è scelto di utilizzare **Styled Components**, che consente di definire i CSS all'interno del codice React, migliorando la leggibilità e semplificando la gestione degli stili. Questa libreria offre anche funzionalità avanzate come la creazione di temi e la gestione delle animazioni, consentendo una personalizzazione flessibile dell'aspetto dell'applicazione. Per gestire lo stato dell'applicazione e le chiamate ai servizi REST, si è scelto di utilizzare **React Query**. Questa libreria semplifica l'interazione con le API, permettendo di eseguire query in modo intuitivo, gestire la cache e gestire lo stato dei dati.

#### 4.1.3 Postgres

Per quanto riguarda la persistenza dei dati, si è optato per l'utilizzo di **PostgreSQL come database**. PostgreSQL è un sistema di gestione di database relazionale affidabile e scalabile, che offre un'ampia gamma di funzionalità. La sua architettura robusta e la sua flessibilità lo rendono una scelta adatta per l'archiviazione dei dati dell'applicazione, garantendo l'affidabilità, l'integrità dei dati e buone performance.

#### 4.1.4 Express

Per lo sviluppo del backend dell'applicazione, si è scelto di utilizzare Express come framework web. Express è un framework veloce e leggero per Node.js che semplifica la gestione delle richieste HTTP e la **creazione di API REST**. Grazie alla sua semplicità e alla vasta comunità di sviluppatori, Express offre un ambiente di sviluppo flessibile e scalabile per gestire le richieste del client e implementare la logica di business dell'applicazione.

### 4.2 Documentazione del backend

**Documentazione delle API** Le API sono state documentate con il tool SwaggerUI che permette di generare una descrizione dettagliata di ogni route del backend e dà anche la possibilità di formattare e inviare richieste HTTP al server. [Link documentazione](#) (Link alla documentazione in cloud nella route /doc).

#### 4.2.1 Tutte le routes del backend

Ratatouille - Documentazione API 0.0.1 OAS3

Software gestionale destinato all'uso nel settore della ristorazione.

Servers  Authorize

**Utente**

- POST /login**
- POST /register**
- POST /utenza/{id\_ristorante}**
- GET /utenti/{id\_ristorante}**
- PUT /utente/{email}**
- DELETE /utente/{email}**
- GET /utente/{id}**
- GET /pw-changed**
- POST /pw-change**

**Ristorante**

- GET /restaurants**
- POST /restaurant/{id}**
- GET /restaurant**
- POST /restaurant**

**Menu**

- GET /categorie/{id\_ristorante}**
- POST /categoria**
- DELETE /categoria/{id\_categoria}**
- PUT /categoria/{id\_categoria}**

**Ordinazione**

- POST /ordina/{id\_ristorante}**
- POST /ordinazioni/evase/**
- GET /ordinazioni/{is\_evase}**
- GET /ordinazione/{id\_ordinazione}**

**Elemento**

- GET /elementi/{id\_categoria}**
- GET /elemento/{id\_elemento}**
- PUT /scambia-elementi/{id\_elemento1}/{id\_elemento2}**
- PUT /elemento/:id\_elemento**
- DELETE /elemento/:id\_elemento**
- POST /elemento**

```

Conto
  GET /conta
Allergene
  POST /allergene
  GET /allergeni/{id_elemento}
  DELETE /allergene/{id_allergene}

```

#### 4.2.2 Esempio di route

In questo esempio viene mostrato come SwaggerUI permette una documentazione dettagliata delle routes, l'invio di richieste HTTP formattate al server e la ricezione delle risposte.

```

POST /utenza/{id_ristorante}

Crea l'account di un utente amministratore

Parameters
Name Description
id_ristorante id del ristorante a cui appartiene l'utente
(path)
1

Request body
application/json

{
  "name": "x",
  "cognome": "y",
  "ruolo": "CAMERIERE",
  "telefono": "3445566778",
  "supervisore": "false",
  "email": "xxxxxx.caio@gmail.com",
  "password": "123"
}

```

Figura 10: Dettaglio su /utenza/{id\_restaurant}.

Code	Details
200	Response body <pre>{   "success": true,   "data": "Registrazione avvenuta con successo" }</pre> Response headers <pre> access-control-allow-origin: * connection: keep-alive content-length: 61 content-type: application/json; charset=utf-8 date: Mon, 01 May 2023 15:18:12 GMT etag: W/3d-1BnixXCG1g-bgIMHvtelWPc5dk8* keep-alive: timeout=5 x-powered-by: Express </pre>

Figura 11: Reale risposta al curl effettuato mediante SwaggerUI.

Responses		Links
Code	Description	
200	Utente registrato con successo	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header</p> <p><a href="#">Example Value</a>   <a href="#">Schema</a></p> <pre>{   "success": true,   "data": "Registrazione avvenuta con successo" }</pre>	
400	Errore durante la registrazione	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header</p> <p><a href="#">Example Value</a>   <a href="#">Schema</a></p> <pre>{   "success": false,   "data": "Errore durante la registrazione" }</pre>	
403	Accesso consentito solo a un admin	No links
	<p>Media type</p> <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">application/json</div> <p>Controls Accept header</p> <p><a href="#">Example Value</a>   <a href="#">Schema</a></p> <pre>{   "success": false,   "data": "Invalid token" }</pre>	

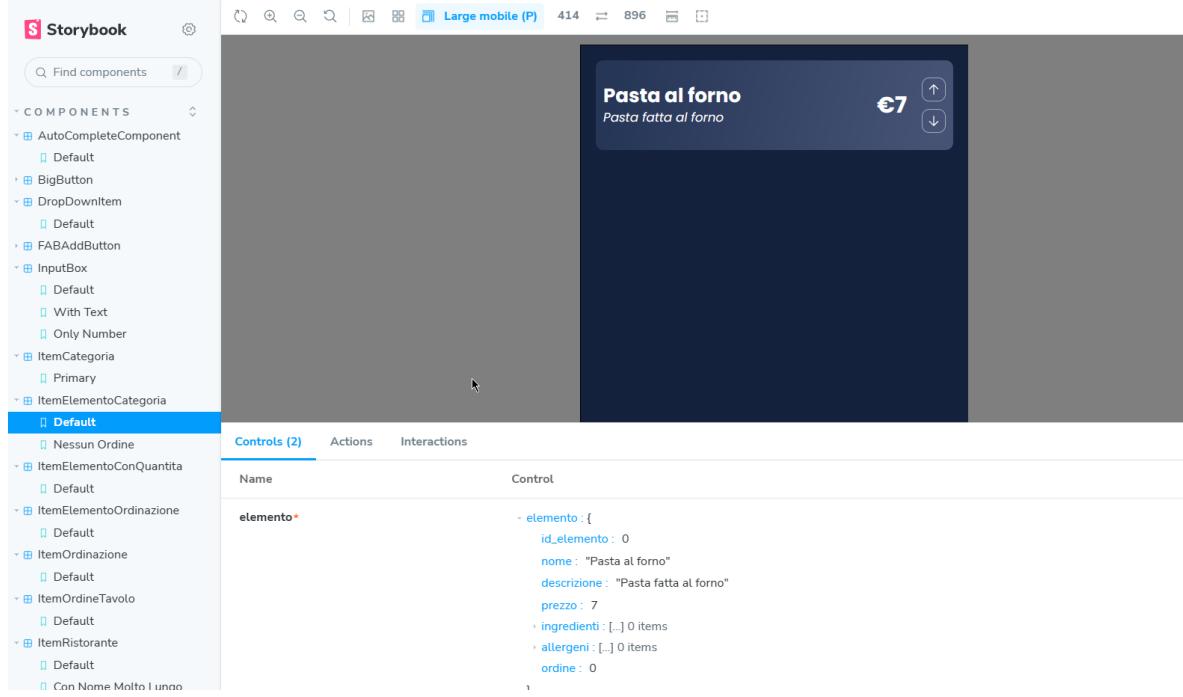
Figura 12: Esempi predefiniti di risposte.



Figura 13: SwaggerUI - Tool di documentazione del backend.

### 4.3 Documentazione del frontend

Il frontend è stato documentato con l'ausilio di Storybook. Storybook è un tool utile per la documentazione e il testing dei componenti di un'applicazione web sviluppata in React. Consente di creare un ambiente di sviluppo in cui è possibile visualizzare tutti i componenti React e generare automaticamente una documentazione completa. La documentazione può essere compilata e rilasciata su un server, rendendola disponibile anche per altri membri del team o per la comunità open source. Storybook riduce il tempo e lo sforzo necessario per lo sviluppo e la documentazione dei componenti React. Ecco un'anteprima di Storybook:



The screenshot shows the Storybook interface. On the left, there is a sidebar titled "Storybook" with a search bar and a "Find components" input field. Below the search bar is a tree view of components under "COMPONENTS". The "Default" component is selected, highlighted with a blue border. Other components listed include AutoCompleteComponent, BigButton, DropDownItem, FABAddButton, InputBox, ItemCategoria, ItemElementoCategoria, ItemElementoConQuantita, ItemElementoOrdinazione, ItemOrdinazione, ItemOrdineTavolo, and ItemRistorante. Under "ItemElementoCategoria", the "Default" component is also selected. To the right of the sidebar is a preview panel titled "Large mobile (P)" showing a card for "Pasta al forno" with a price of €7. Below the preview panel is a table titled "Controls (2)" with two rows. The first row has columns "Name" and "Control". The second row contains the value "elemento" followed by a JSON object representing the component's state.

Name	Control
elemento*	- elemento : { id_elemento : 0 nome : "Pasta al forno" descrizione : "Pasta fatta al forno" prezzo : 7 ingredienti : [...] 0 items allergeni : [...] 0 items ordine : 0

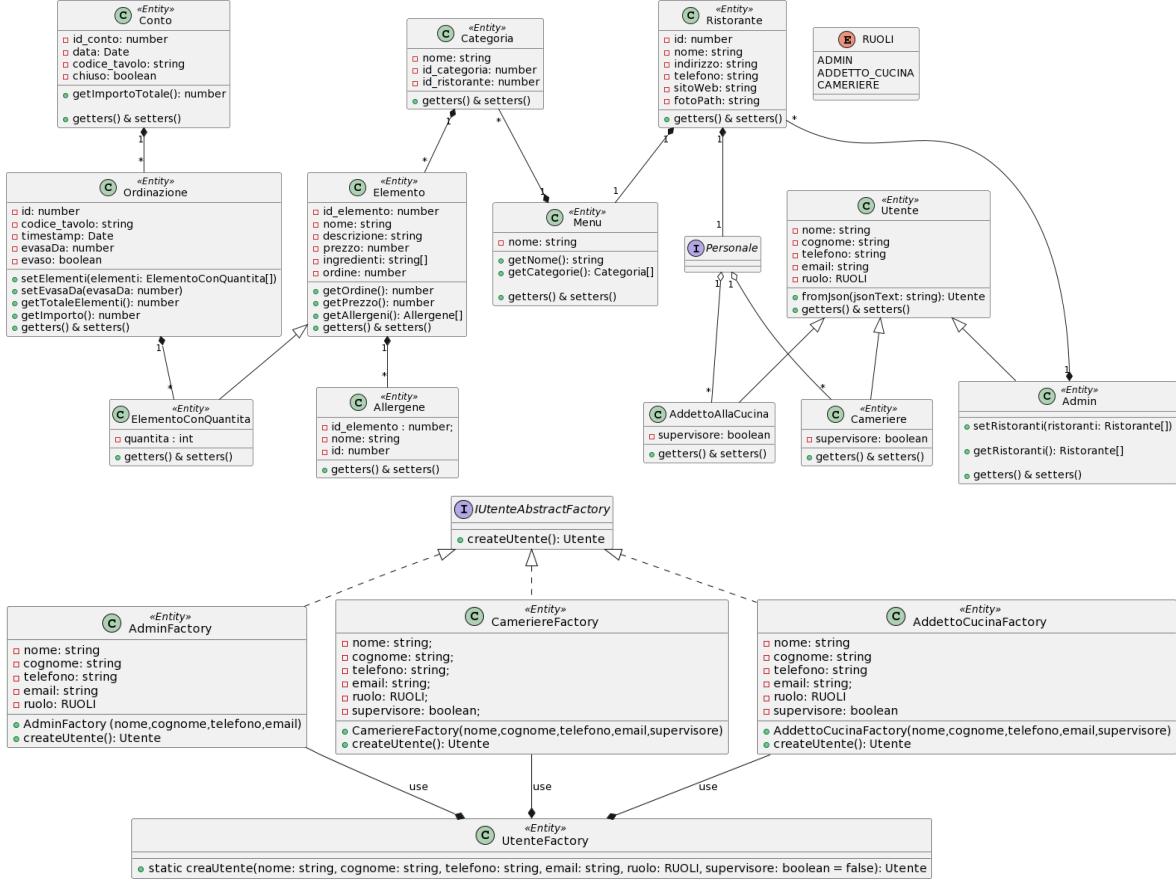
[Link documentazione](#) (Link alla documentazione in cloud).



Figura 14: StoryBook - Tool di Documentazione del frontend

## 5 Diagramma delle classi di design

### 5.1 Le Entities

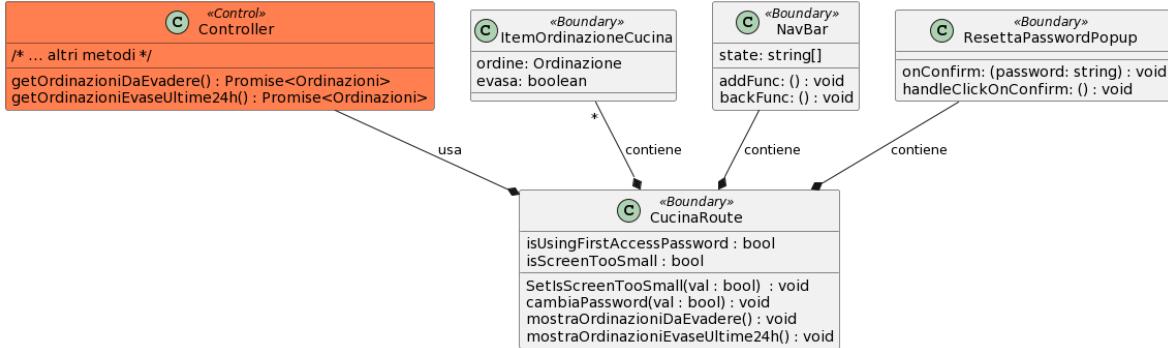


#### 5.1.1 Utilizzo del design pattern Factory Method

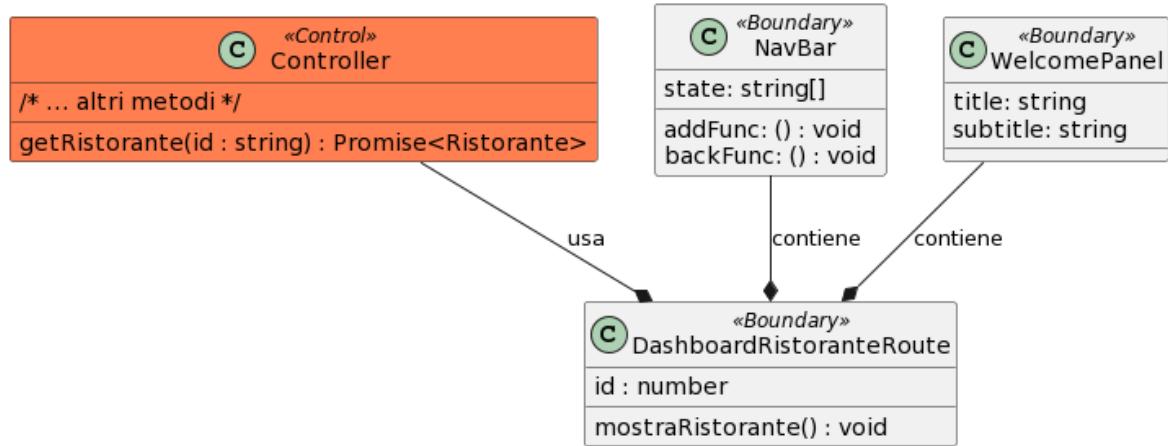
E' stato utilizzato il design pattern "Factory Method" per istanziare l'utente correttamente. Il design pattern Factory Method permette di creare oggetti senza bisogno di dover specificare la loro classe. Ciò significa che il codice interagisce esclusivamente con l'interfaccia risultante o la classe astratta, in modo che funzioni con qualsiasi classe che implementi l'interfaccia o che estenda la classe *IUtenteAbstractFactory*. Per istanziare un utente basterà chiamare il metodo statico *UtenteFactory.creaUtente(nome, cognome...)* con i relativi parametri. Il Factory Method provvederà a gestire la logica di creazione dell'utente e restituirà correttamente un *Utente* di sottoclassi *Admin*, *Cameriere* o *AddettoAllaCucina*. Strutturando il codice secondo questo pattern è possibile anticipare il cambiamento delle specifiche e aggiungere nuovi tipi di utenti senza dover rivoluzionare il class diagram.

## 5.2 I Boundaries

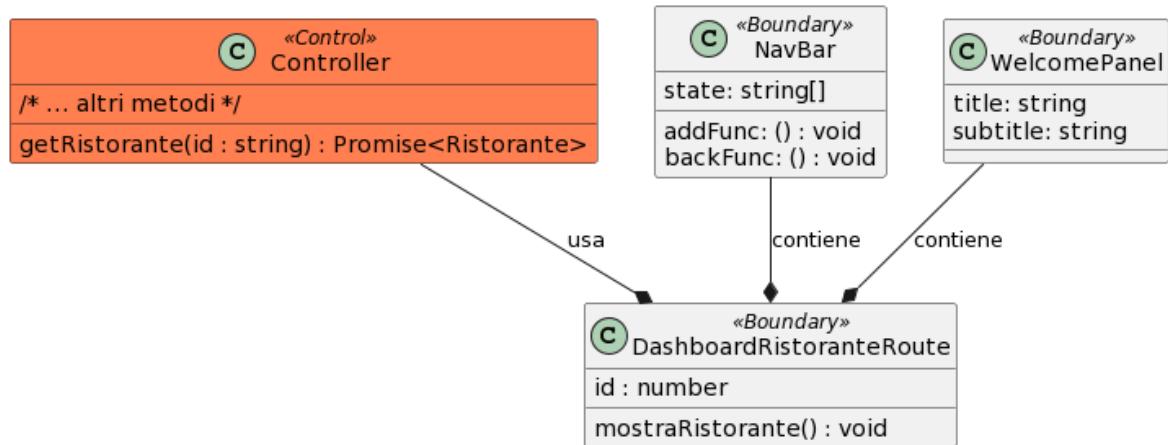
### 5.2.1 Cucina



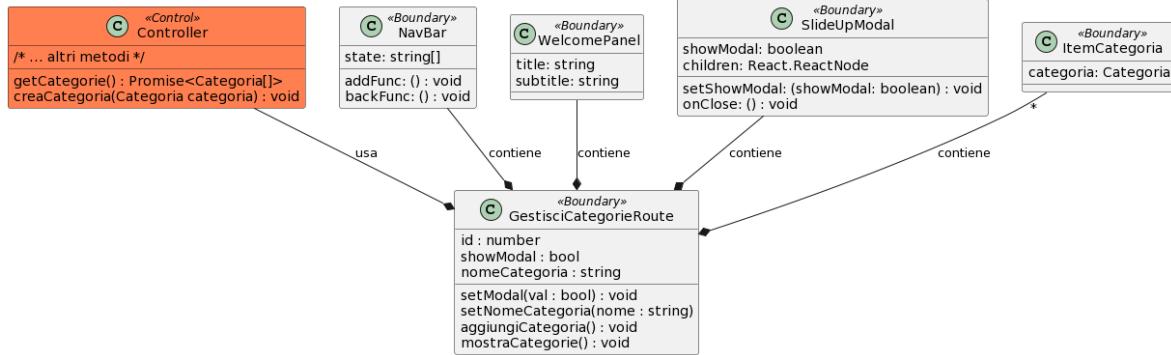
### 5.2.2 Dashboard ristorante



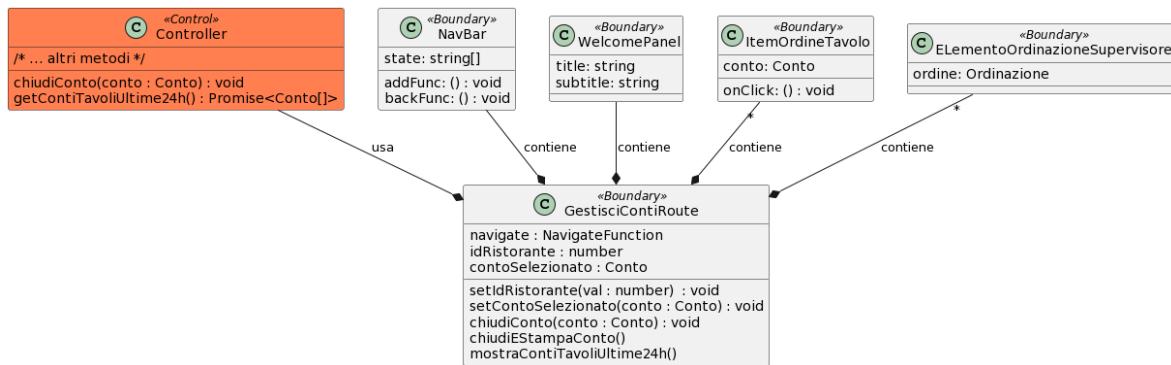
### 5.2.3 Dashboard supervisore



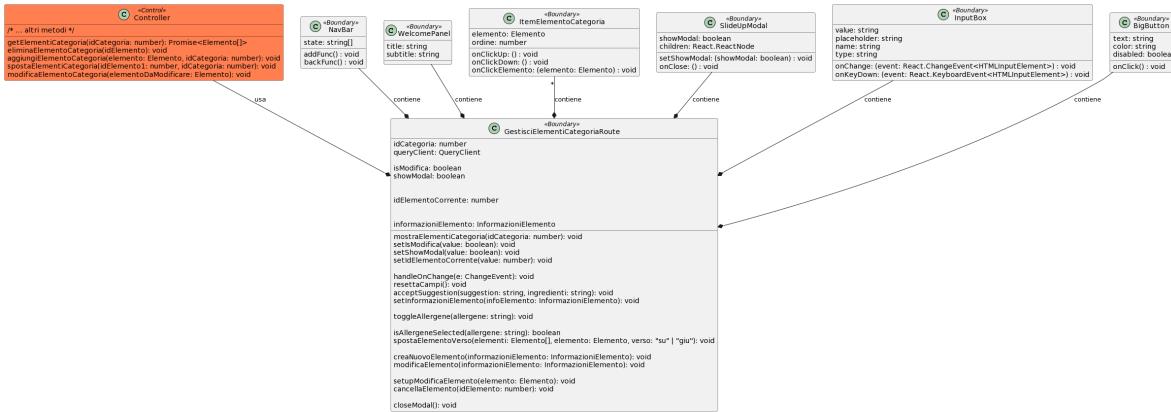
## 5.2.4 Gestisci categorie



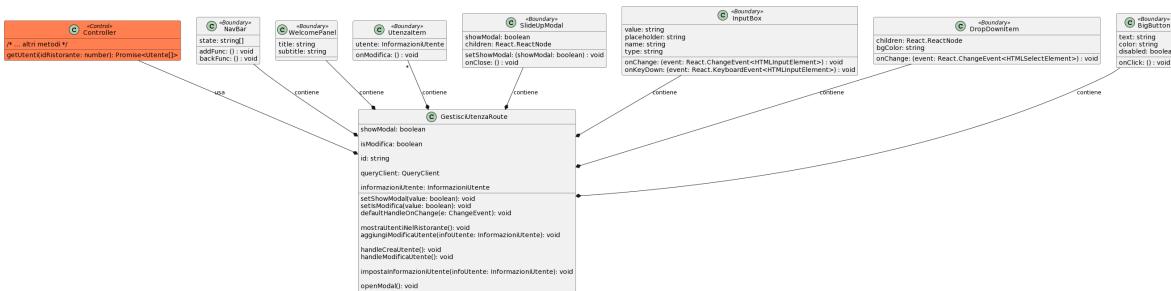
## 5.2.5 Gestisci conti



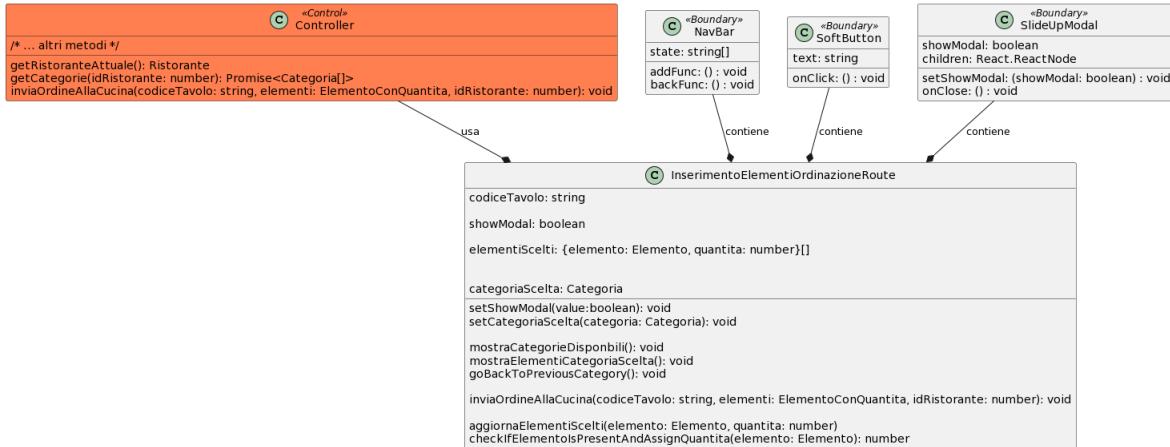
## 5.2.6 Gestisci elementi di una categoria



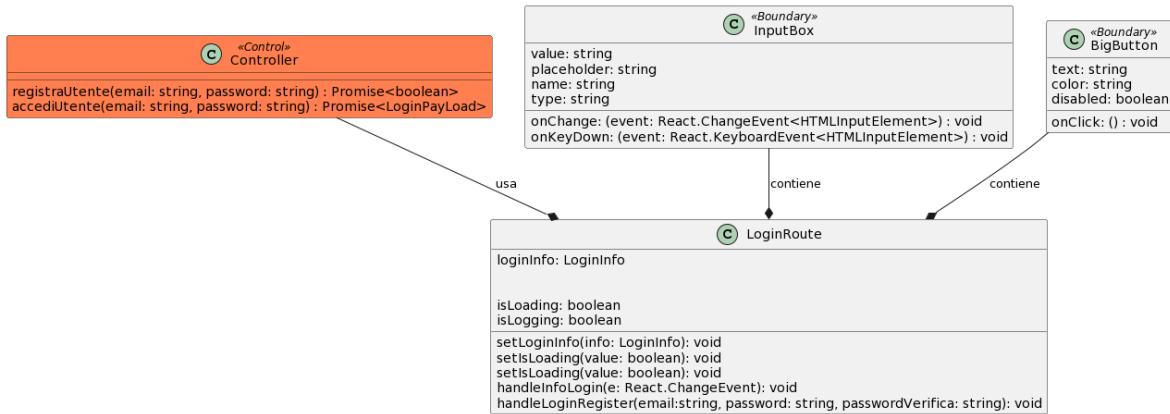
## 5.2.7 Gestisci utenza



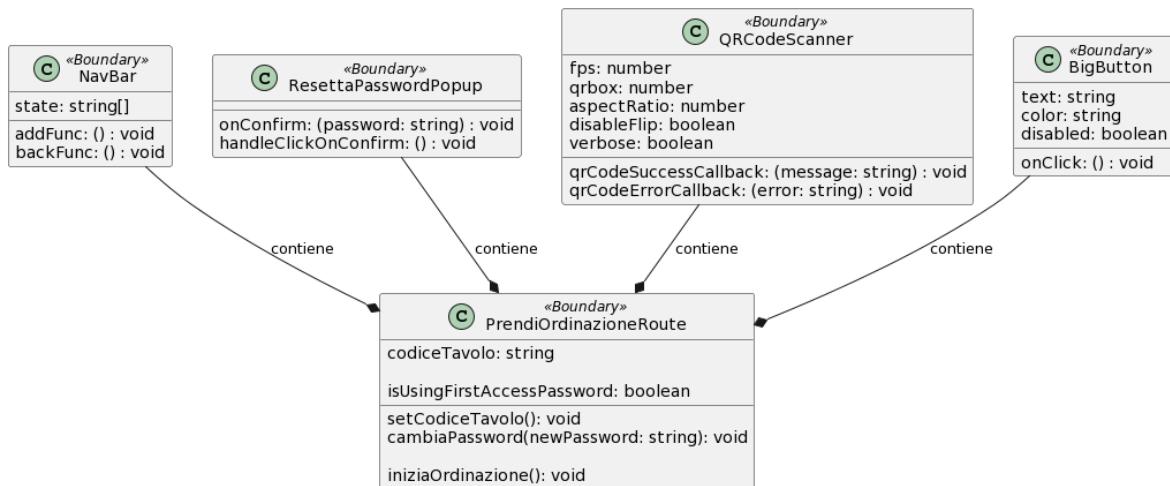
### 5.2.8 Inserimento elementi di un'ordinazione



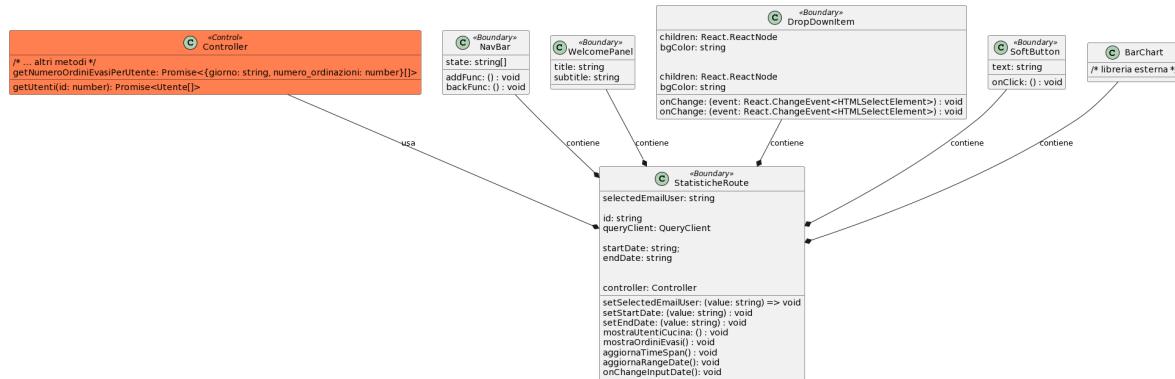
### 5.2.9 Login e registrazione



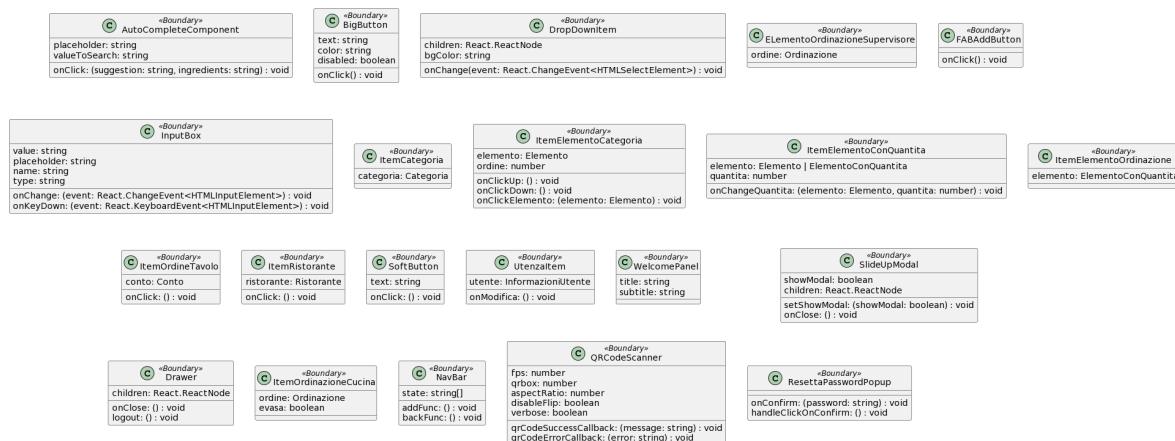
### 5.2.10 Ordina



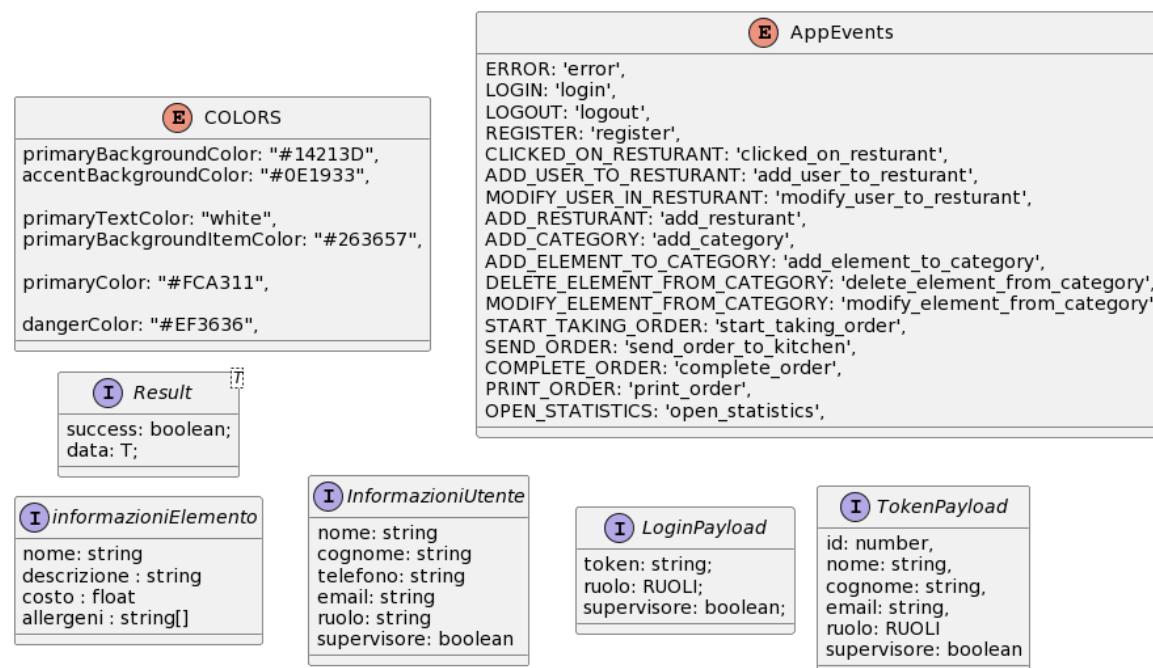
### 5.2.11 Statistiche



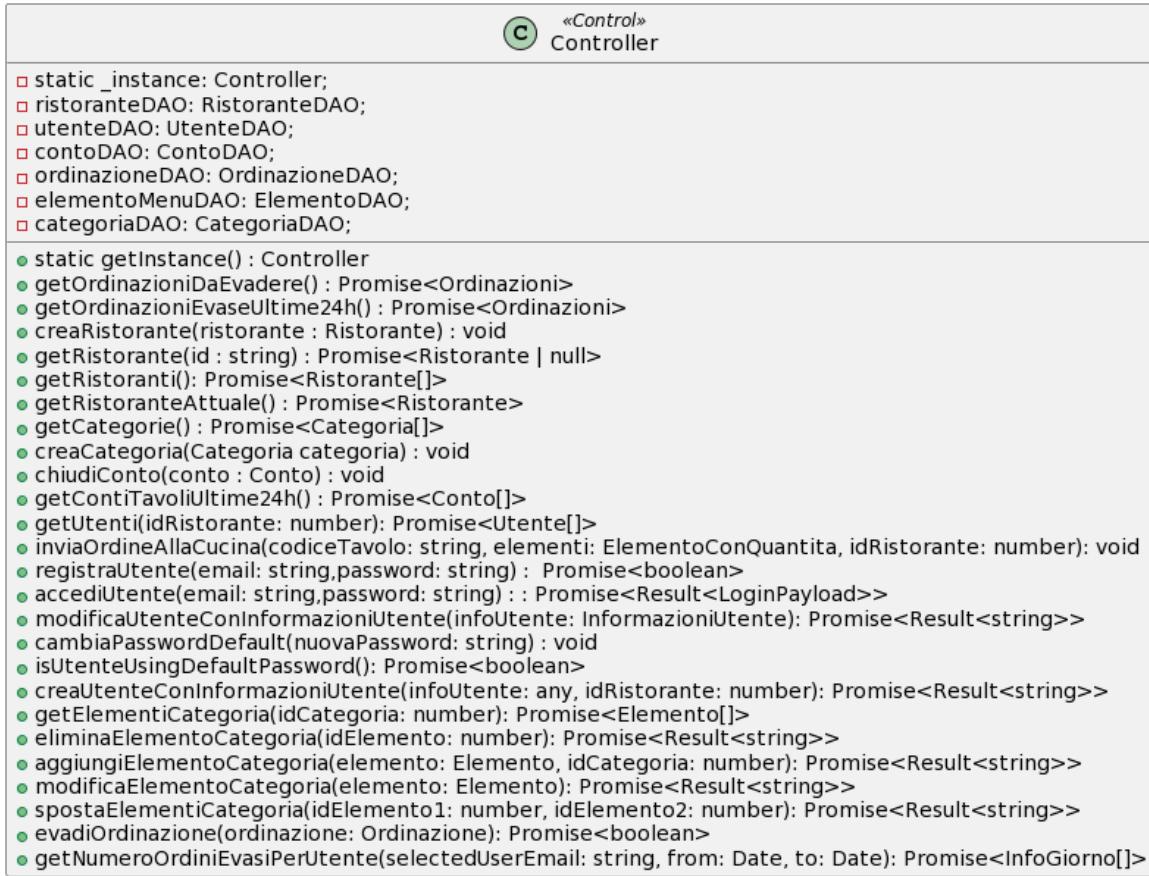
### 5.2.12 Tutti i components



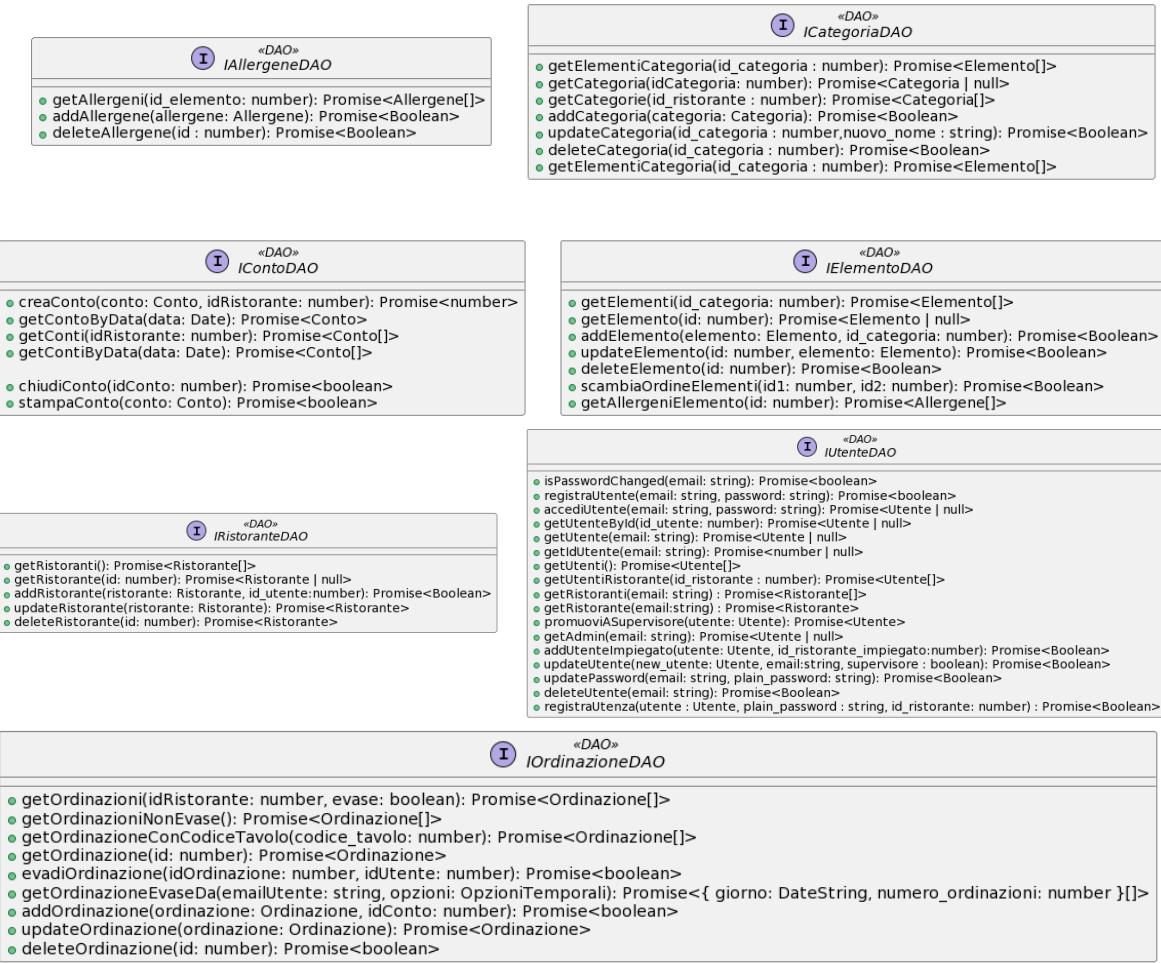
### 5.2.13 Interfacce e enums di utilità



### 5.3 Il Controller

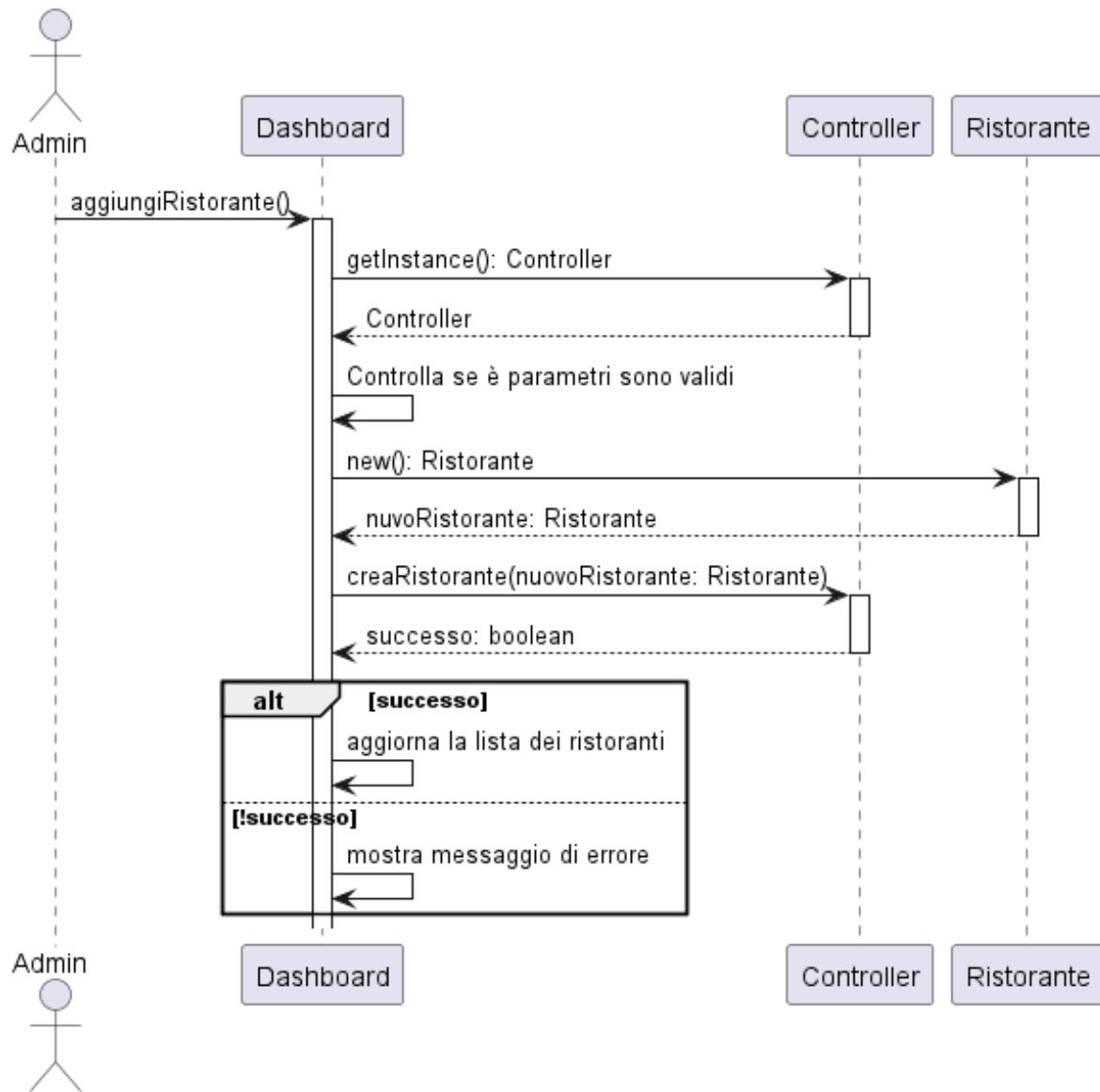


## 5.4 DAOs

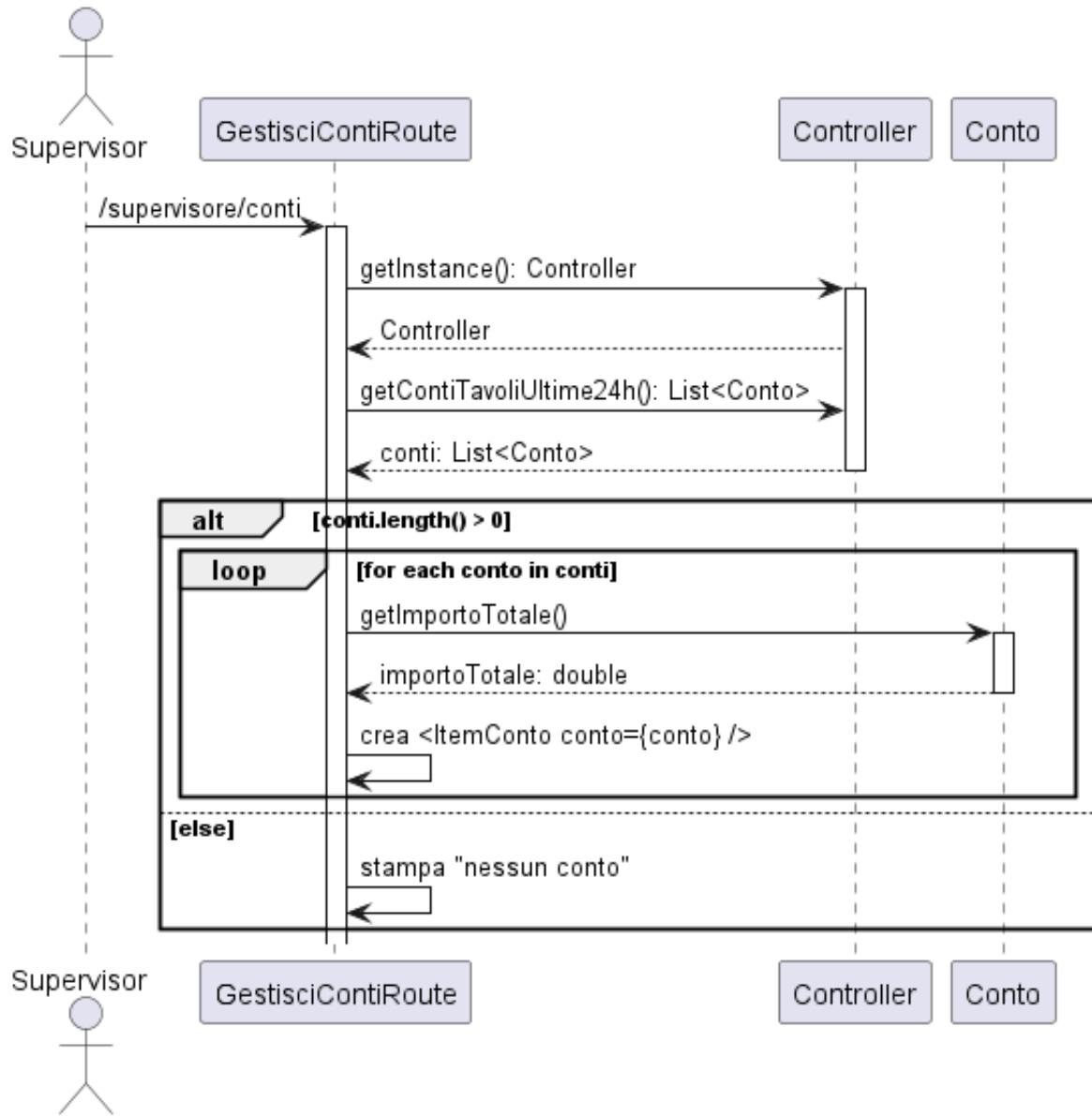


## 6 Diagrammi di sequenza di design

### 6.1 aggiungiRistorante



## 6.2 getContiUltime24h



### III Testing e valutazione sul campo dell'usabilità

## 7 Codice xUnit Black Box per Unit Testing

Al fine di garantire una maggiore affidabilità del software è stato effettuato unit testing su 4 funzioni non banali con almeno 2 parametri.

### 7.1 Funzione addElemento

La funzione **addElemento** aggiunge un elemento a una categoria del menu. I primi due parametri sono destinati alla query all'API e il terzo parametro è il token JWT. Ecco la signature del metodo:

```
async addElemento(elemento: Elemento, idCategoria: number, token?: string): Promise<Result<string>>
```

#### 7.1.1 Strategia di testing Black Box

Sono state individuate le classi di equivalenza come oggetto del test.

Nome classe equivalenza	Nome parametro	Insieme dei valori	Output
elementoCE1	elemento	{nome vuoto}	Errore
elementoCE2	elemento	{quantità <=0}	Errore
elementoCE3	elemento	{valido}	Successo
idCategoriaCE1	idCategoria	{inesistente nel db}	Errore
idCategoriaCE3	idCategoria	{esistente nel db}	Successo
tokenCE1	token	{undefined}	Errore
tokenCE2	token	{valido}	Autenticato
tokenCE3	token	{non valido}	Non autenticato

E' stata utilizzata la strategia SECT (Strong Equivalence Class Testing) per implementare i metodi riportati nel codice xUnit Black Box di seguito.

### 7.1.2 Codice xUnit Black Box

```
const email = 'mario.rossi@gmail.com';
const password = 'mario';
const token = (await utenteDAO.accediUtente(email, password)).data.token;

describe('addElement()', () => {
    it("dovrebbe ritornare true se l'elemento è stato registrato con successo", async () => {
        const elemento = new Elemento(`test${randomInt(8000)}`,
            "Descrizione",
            2,
            {
                ingredienti: [],
                allergeni: [],
                ordine: 0
            }
        );
        const result = await elementoDAO.addElement(elemento,1,token);
        expect(result.success).toBe(true);
        expect(result.data).toBeTypeOf('string');
    });

    it("dovrebbe ritornare false se il nome dell'elemento è ''", async () => {
        const elemento = new Elemento("", "Descrizione",
            2,
            {
                ingredienti: [],
                allergeni: [],
                ordine: 0
            }
        );
        const result = await elementoDAO.addElement(elemento,1,token);
        expect(result.success).toBe(false);
        expect(result.data).toBeTypeOf('string');
    });

    it("dovrebbe ritornare false se il prezzo è negativo", async () => {
        const elemento = new Elemento("prova", "Descrizione",
            -1,
            {
                ingredienti: [],
                allergeni: [],
                ordine: 0
            }
        );
        const result = await elementoDAO.addElement(elemento,1,token);
        expect(result.success).toBe(false);
        expect(result.data).toBeTypeOf('string');
    });
});
```

## 7.2 Funzione scambiaElementi

La funzione **scambiaElementi** ha due parametri destinati alla query dell'API e un terzo parametro che è il token JWT. Ecco la signature del metodo:

```
async scambiaElementi(idElemento1: number, idElemento2: number, token?: string): Promise<Result<string>>
```

## 7.3 Strategia di testing Black Box

Sono state individuate le classi di equivalenza come oggetto del test.

Nome classe equivalenza	Nome parametro	Insieme dei valori	Output
idElemento1CE1	idElemento1	{id > 0}	Successo
idElemento1CE2	idElemento1	{id ≤ 0}	Errore
idElemento2CE1	idElemento2	{id > 0}	Successo
idElemento2CE2	idElemento2	{id ≤ 0}	Errore
tokenCE1	token	{undefined}	Errore
tokenCE2	token	{valido}	Autenticato
tokenCE3	token	{non valido}	Non autenticato

E' stata utilizzata la strategia SECT (Strong Equivalence Class Testing) per implementare i metodi riportati nel codice xUnit Black Box di seguito.

### 7.3.1 Codice xUnit Black Box

```
describe('scambiaElementi()', () => [  
  it("Lo scambio di due elementi esistenti dovrebbe ritornare true", async () => {  
    const result = await elementoDAO.scambiaElementi(1,2,token);  
    expect(result.success).toBe(true);  
    expect(result.data).toBeTypeOf('string');  
  });  
  
  it("Lo scambio dello stesso elemento dovrebbe ritornare un messaggio di alert", async () => {  
    const result = await elementoDAO.scambiaElementi(1,1,token);  
    expect(result.success).toBe(true);  
    expect(result.data).toBeTypeOf('string');  
    expect(result.data).toBe('Non ha senso scambiare lo stesso elemento');  
  });  
  
  it("Lo scambio elementi che non esistono non solleva errore e ritorna false", async () => {  
    const result = await elementoDAO.scambiaElementi(-2,-1,token);  
    expect(result.success).toBe(false);  
    expect(result.data).toBeTypeOf('string');  
  });  
];
```

## 7.4 Funzione registraUtente

La funzione **registraUtente** ha due parametri e se non esiste un utente con la stessa email lo crea. Ecco la signature del metodo:

```
async registraUtente(email: string, password: string): Promise<boolean> {
```

### 7.4.1 Strategia di testing Black Box

Sono state individuate le classi di equivalenza come oggetto del test.

Nome classe equivalenza	Nome parametro	Insieme dei valori	Output
emailCE1	email	{stringa vuota}	Errore
emailCE2	email	{stringa valida}	Esito autenticazione
emailCE3	email	{email esistente}	Esito negativo autenticazione
passwordCE1	password	{stringa vuota}	Errore
passwordCE2	password	{stringa valida}	Esito autenticazione

E' stata utilizzata la strategia SECT (Strong Equivalence Class Testing) per implementare i metodi riportati nel codice xUnit Black Box di seguito.

#### 7.4.2 Codice xUnit Black Box

```
describe('registraUtente()', () => {  
  
    it("dovrebbe ritornare false se l'email o la password sono vuoti ('')", async () => {  
        const email = '';  
        const password = '';  
  
        const response = await utenteDAO.registraUtente(email, password);  
  
        expect(response).toBe(false)  
    });  
  
    it("dovrebbe ritornare true se l'utente è stato registrato con successo", async () => {  
        const email = `test${randomInt(8000)}@test.com`;  
        const password = 'password';  
  
        const result = await utenteDAO.registraUtente(email, password);  
  
        expect(result).toBe(true)  
    });  
  
    it("dovrebbe ritornare false se l'utente non è stato registrato con successo, perchè già registrato", async () => {  
        const email = 'mario.rossi@gmail.com';  
        const password = 'mario';  
  
        const result = await utenteDAO.registraUtente(email, password);  
  
        expect(result).toBe(false)  
    });  
});
```

## 7.5 Funzione accediUtente

La funzione `accediUtente` ha due parametri, autentica l'utente e ritorna un payload. Ecco la signature del metodo:

```
async accediUtente(email: string, password: string): Promise<Result<LoginPayload>>
```

### 7.5.1 Strategia di testing Black Box

Sono state individuate le classi di equivalenza come oggetto del test.

Nome classe equivalenza	Nome parametro	Insieme dei valori	Output
emailCE1	email	{stringa vuota}	Errore
emailCE2	email	{stringa valida}	Esito autenticazione dell'utente
passwordCE1	password	{stringa vuota}	Errore
passwordCE2	password	{stringa valida}	Esito autenticazione dell'utente

E' stata utilizzata la strategia SECT (Strong Equivalence Class Testing) per implementare i metodi riportati nel codice xUnit Black Box di seguito.

### 7.5.2 Codice xUnit Black Box

```
describe('accediUtente()', () => {

    it("dovrebbe ritornare un oggetto di tipo Result<string> se l'utente è stato loggato con successo", async () => {
        const email = "mario.rossi@gmail.com"
        const password = "mario"

        const result = await utenteDAO.accediUtente(email, password);

        expect(result.success).toBe(true);
        expect(result.data).toBeTypeOf('object');
    });

    it("dovrebbe ritornare un oggetto di tipo Result<string> se l'utente non è stato loggato con successo", async () => {
        const email = "mario.rossi@gmail.com"
        const password = "password_errata"

        const result = await utenteDAO.accediUtente(email, password);

        expect(result.success).toBe(false);
        expect(result.data).toBeTypeOf('object');
    });

    it("dovrebbe ritornare un oggetto di tipo Result<string> se l'email o la password sono vuoti ('')", async () => {
        const email = "";
        const password = "";

        const result = await utenteDAO.accediUtente(email, password);

        expect(result.success).toBe(false);
        expect(result.data).toBeTypeOf('object');
    });
});
```

## 8 Valutazione dell'usabilità sul campo

### 8.1 Test di valutazione finale

Durante lo sviluppo del software sono state corrette alcune problematiche che influenzavano in modo negativo l'esperienza utente. Grazie a un test di valutazione finale, effettuato su un gruppo di testers, è stato possibile ottenere un feedback positivo dell'applicazione. Ecco i risultati del test:

	1	2	3	4	5	6	7
Tester 1	✓	✓	✓	✓	✓	✓	✓
Tester 2	✓	✓	✓	✓	✓	✓	✓
Tester 3	✓	✓	✓	✓	✓	✓	✓
Tester 4	✓	✓	✓	✓	✓	⚠️	✓

**Leggenda:**  
✓ Successo +1  
✗ Fallimento -1  
⚠️ Successi Parziali +0.5

Il punteggio ottenuto è migliorato rispetto al precedente test (Vedi Figura 2 nella sezione 1.6).

### 8.2 Analisi mediante file di log

Il sistema Ratatouille23 raccoglie dati sugli utenti a scopo di analisi. Per raccogliere dati sugli utenti è stato utilizzata la piattaforma Firebase. Firebase è una piattaforma di sviluppo mobile e web, acquisita da Google. Essa fornisce una vasta gamma di servizi, tra cui l'autenticazione degli utenti, il database in tempo reale, l'hosting dei siti web, il cloud storage, i servizi di messaggistica, le funzionalità di analisi come Google Analytics. Firebase e Google Analytics sono strettamente integrati e consentono di tracciare gli eventi degli utenti e le attività all'interno di un'applicazione mobile o web. In particolare, Firebase fornisce la possibilità di inviare eventi personalizzati ad Analytics, in modo da tenere traccia dell'utilizzo dell'applicazione e degli eventuali problemi riscontrati dagli utenti.



### 8.3 Integrazione con SDK di Firebase

Integrare Firebase nel progetto è stato estremamente semplice grazie alla ottima documentazione. Sono state necessarie poche modifiche per aggiungere il servizio nell'applicativo. Firebase è in grado di tener traccia degli eventi generati dall'utente. Gli eventi sono conservati in un oggetto detto *AppEvents*. Quando si vuole registrare un evento viene invocata la funzione *logEvent* di cui è stato realizzato un wrapper che semplifica la scelta dei tipi di eventi da inviare con il relativo payload opzionale.

```
32 export const AppEvents = {
33   ERROR: 'error',
34   LOGIN: 'login',
35   LOGOUT: 'logout',
36   REGISTER: 'register',
37   CLICKED_ON_RESTAURANT: 'clicked_on_restaurant',
38   ADD_USER_TO_RESTAURANT: 'add_user_to_restaurant',
39   MODIFY_USER_IN_RESTAURANT: 'modify_user_to_restaurant',
40   ADD_RESTAURANT: 'add_restaurant',
41   ADD_CATEGORY: 'add_category',
42   ADD_ELEMENT_TO_CATEGORY: 'add_element_to_category',
43   DELETE_ELEMENT_FROM_CATEGORY: 'delete_element_from_category',
44   MODIFY_ELEMENT_FROM_CATEGORY: 'modify_element_from_category',
45   START_TAKING_ORDER: 'start_taking_order',
46   SEND_ORDER: 'send_order_to_kitchen',
47   COMPLETE_ORDER: 'complete_order',
48   PRINT_ORDER: 'print_order',
49   OPEN_STATISTICS: 'open_statistics',
50 } as const;
```

Figura 16: Oggetto con tutti i tipi di evento registrati.

```
1 import { AppEvents } from './utils/constants';
2 import { initializeApp } from "firebase/app";
3 import { getAnalytics, logEvent } from "firebase/analytics";
4
5 const firebaseConfig = {
6   apiKey: "",
7   authDomain: "ratatouille-app-2023.firebaseio.com",
8   projectId: "ratatouille-app-2023",
9   storageBucket: "ratatouille-app-2023.appspot.com",
10  messagingSenderId: "",
11  appId: "",
12  measurementId: ""
13};
14
15 type AppEventsType = typeof AppEvents[keyof typeof AppEvents];
16
17 const logEventToFirebase = (eventType: AppEventsType, payload?: any) => {
18   logEvent(analytics, eventType.toString(), payload);
19 }
20
21 // Initialize Firebase
22 const app = initializeApp(firebaseConfig);
23 const analytics = getAnalytics(app);
24
25 export {
26   analytics,
27   logEventToFirebase
28 }
```

Figura 17: API keys di Firebase e wrapper di *logEvents*.

## 8.4 Analisi dei dati ottenuti con Google Analytics

Effettuando il login sulla piattaforma Firebase è possibile consultare le statistiche degli utenti del software Ratatouille<sup>23</sup>. Ecco un'anteprima delle informazioni ottenute dagli utenti.

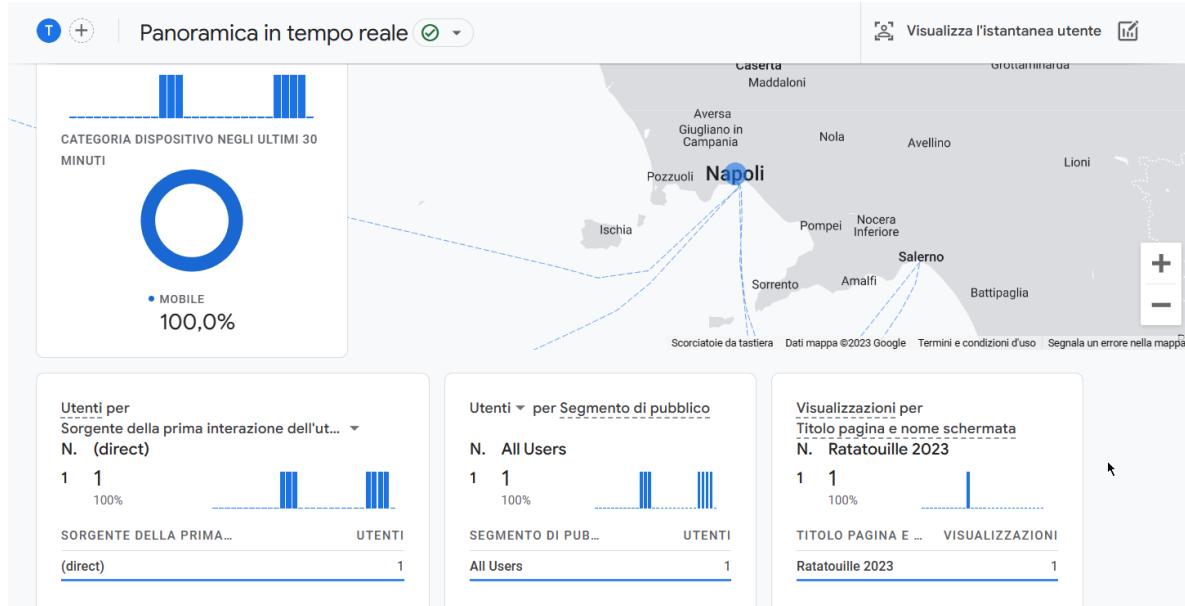
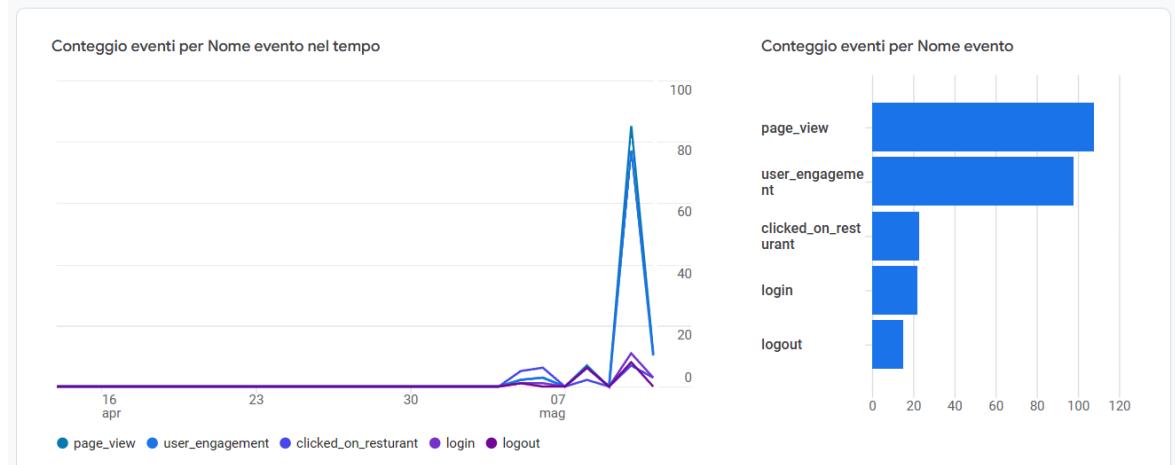


Figura 18: Panoramica generale con localizzazione degli utenti.



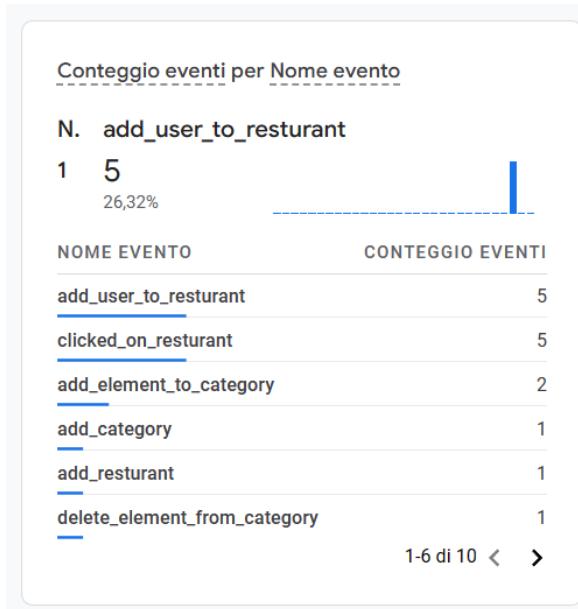


Figura 19: Conteggio degli eventi

## 8.5 File di log

Firebase permette di scaricare un resoconto (file di log) delle statistiche degli utenti. Ecco il file di log con tutte le informazioni ottenute dagli utenti [Link al file di log](#)

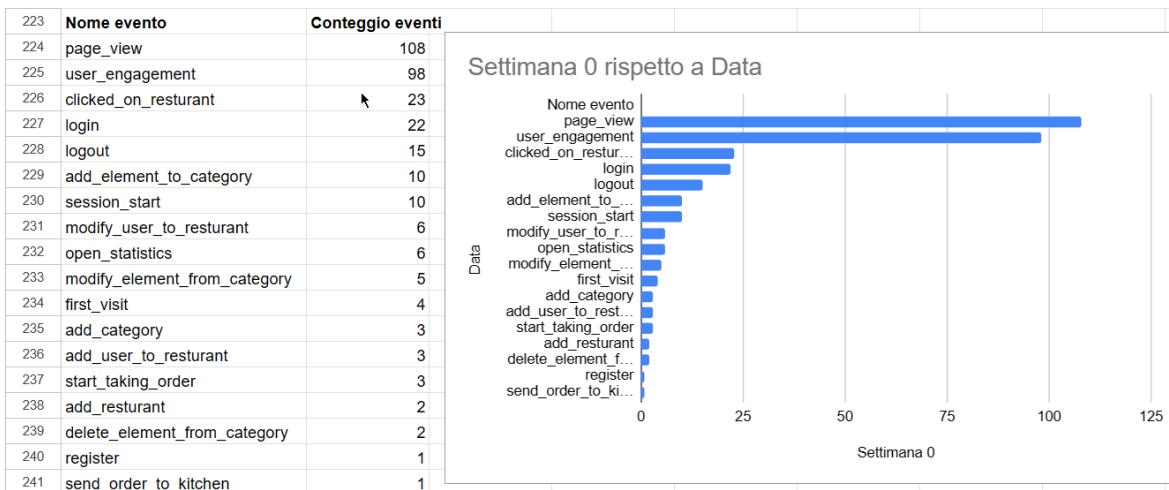


Figura 20: Visualizzazione del file di log CSV in Google Sheets