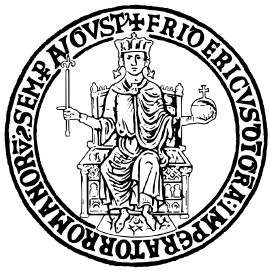
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Ingegneria del software

Ratatouille23

Professori
S. D. Martino
F. cutugno
L. L. L. starace
M. grazioso

Candidato
Paolo Cammardella
N86/3043

Anno Accademico 2022–2023

Indice

0 Glossario

Qui sono racchiuse tutte le definizioni e gli acronimi che sono stati utilizzati nella documentazione.

0.1 Definizioni

Scalabilità si riferisce alla capacità di un sistema di aumentare o diminuire le risorse a seconda delle necessità.

Usabilità l'usabilità di un prodotto è il grado con cui esso può essere usato da specificati utenti per raggiungere specificati obiettivi con efficacia, efficienza e soddisfazione in uno specificato contesto d'uso.

Mock-up Un mock-up, è una realizzazione a scopo illustrativo o meramente espositivo di un oggetto o un sistema, senza le complete funzioni dell'originale. Un mockup può rappresentare la totalità o solo una parte dell'originale di riferimento (già esistente o in fase di progetto), essere in scala reale oppure variata.

Class diagram I class diagram (diagrammi delle classi, in italiano) sono uno dei tipi di diagrammi che possono comparire in un modello UML.
In termini generali, consentono di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni fra questi tipi. Gli strumenti concettuali utilizzati sono il concetto di classe del paradigma object-oriented e altri correlati.

Sequence diagram Un Sequence Diagram (Diagramma di sequenza, in italiano) è un diagramma previsto dall'UML utilizzato per descrivere uno scenario.

Scenario Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate; in pratica nel diagramma non compaiono scelte, né flussi alternativi.

State chart diagram Uno state chart (diagramma di stato, in italiano) è un tipo di diagramma usato in informatica per descrivere il comportamento dei sistemi, il quale viene analizzato e rappresentato tramite una serie di eventi che potrebbero accadere per ciascuno stato. Per poter essere realizzato, il sistema deve essere composto da un numero finito di stati.

Activity diagram Un activity diagram (diagramma di attività in italiano) è un tipo di diagramma che permette di descrivere un processo attraverso dei grafi in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite.

Persona Una persona in UX design è un identikit vero e proprio di un utente (fittizio) ideale, che viene utilizzato per identificare una fascia d'utenti, i quali esprimono le loro esigenze, comportamenti ed interessi.

White-Box Modalità di testing in cui un metodo viene testato in base al codice che contiene.

Black-Box Modalità di testing in cui un'unità viene testata in base ai requisiti.

Framework È un'architettura logica di supporto sul quale un software può essere progettato e realizzato.

Back-end Rappresenta la parte che permette l'effettivo funzionamento di queste interazioni.

Front-end Rappresenta la parte che interagisce con l'utente finale.

Android Sistema operativo realizzato da Google.

Server Programma che ha il compito di offrire servizi ai client richiedenti.

Tabella Cockburn Tabelle utilizzate per la rappresentazione dei casi d'uso.

Use case Rappresenta una funzione o servizio offerto dal sistema ad uno o più attori.

Docker È un popolare software libero progettato per eseguire processi informatici in ambienti isolabili, minimali e facilmente distribuibili chiamati container Linux, con l'obiettivo di semplificare i processi di deployment di applicazioni software.

0.2 Acronimi

JWT Acronimo di JSON Web Token, è un sistema di cifratura e di contatto in formato JSON per lo scambio di informazioni tra i vari servizi di un server.

AWS Amazon Web Services è un'azienda statunitense di proprietà del gruppo Amazon, che fornisce servizi di cloud computing su un'omonima piattaforma *on demand*.

EC2 Uno dei servizi di cloud offerti da AWS.

API Un API (Application Program Interface), in un programma informatico, in italiano "interfaccia di programmazione dell'applicazione", si indica un insieme di procedure atte a risolvere uno specifico problema di comunicazione tra diversi computer, tra diversi software, tra diversi componenti di software. Spesso tale termine designa le librerie software di un linguaggio di programmazione.

HTTP Sta per HyperText Transfer Protocol, è un protocollo a livello applicativo usato come principale sistema per la trasimissione di informazioni via web.

ISO/IEC ISO e IEC cooperano strettamente per creare uno standard internazionale sviluppato specificatamente per la gestione dei servizi IT.

GDPR General Data Protection Regulation, è un regolamento, sviluppato dalla comunità europea, che disciplina il trattamento dei dati personali da parte di aziende.

MVP Model View Presenter è un design pattern, principalmente utilizzato per software Android.

UI User Interface è il termine inglese usato per indicare l'interfaccia grafica.

DTO Data Transfer Object è un design pattern per trasferire dati da una parte back-end a quella front-end (ove necessario).

SECT È una specifica modalità di testing. Consiste nell'effettuare i test tramite prodotto cartesiano dei parametri.

WECT È una specifica modalità di testing. Consiste nell'effettuare il minimo numero di test case che ricoprono tutte le classi d'equivalenza valide e uno per quelle non valide.

1 Introduzione

Ratatouille23™ è un software sviluppato e progettato dal gruppo INGSW2223_N_46 per conto della società SoftEngUniNA™.
Gli sviluppatori si prendono carico della verifica dei moduli software necessari al corretto funzionamento di tale sistema. Ratatouille23™ è un software che offre il supporto alla gestione di attività di ristorazione.
Il sistema consiste in un'applicazione performante e affidabile, attraverso la quale gli utenti possono fruire delle funzionalità del sistema in modo intuitivo, rapido e piacevole ed una parte back-end che garantisce la comunicazione veloce ed affidabile tra i client.

1.1 Cosa contiene questa documentazione

Questa documentazione contiene tutte le informazioni necessarie a comprendere il funzionamento del software Ratatouille23™

- Documento dei Requisiti Software.
- Documento di Design del sistema.
- Definizione di un piano di testing e valutazione sul campo dell'usabilità.

1.2 Tecnologie utilizzate

Lo sviluppo del software Ratatouille23™ è avvenuto tramite:

- **Client**:
  - Android con linguaggio object oriented (Java).
  - Retrofit: richieste HTTP lato android.
- **Server**:
  - JWT
  - Framework Java Spring boot.
  - RestTemplate.
- **Database**: PostgreSQL.
- **Testing**: Suite JUnit.

Il tutto è affiancato tecnologie allo stato dell'arte quali *Docker* per separare in *container* back-end e database, e servizi di *Cloud Computing* come *AWS*, al fine di massimizzare la scalabilità del sistema in vista di un possibile repentino aumento del numero degli utenti nelle fasi iniziali di rilascio al pubblico.

2 Analisi dei requisiti

In questa sezione andremo a descrivere il modello funzionale del software partendo da un analisi dei casi d'uso assegnati per l'applicativo, e proseguendo poi con tabelle cockburn e mock-up.

2.1 Requisiti del software

Lo sviluppo di Ratatouille23™ in parte è stato scandito dalla presenza di requisiti imposti dal cliente per il corretto funzionamento del software, in parte da requisiti tecnologici.

2.1.1 Requisiti funzionali

Il sistema offre le seguenti funzionalità:

1. La possibilità da parte di un amministratore di poter creare utenze per i propri dipendenti (e.g. addetti alla sala, addetti alla cucina, supervisori) con nome utente scelto dall'aministratore ed una password di default.
2. La possibilità da parte di un amministratore (o un supervisore) di personalizzare il menù dell'attività di ristorazione. In particolare, la possibilità di riordinare il menù, creare e/o eliminare elementi dal menù, caratterizzandoli tramite:
   - *nome.*
   - *descrizione.*
   - *elenco di allergeni comuni.*
   - *prezzo.*

   In fase di creazione di un determinato piatto, è disponibile, utilizzando l'apposito tasto per la ricerca, l'autocompletamento di alcuni prodotti (e.g.: bibite o preconfezionati).
3. La possibilità da parte di un addetto alla sala di poter registrare ordinazioni indicando l'identificativo del tavolo e gli elementi del menù (già presenti) desiderati.
4. Un supervisore o un amministratore può inserire nel sistema degli avvisi (chiamati notifiche), che possono essere visualizzati da tutti i dipendenti. Ciascun dipendente può poi , qualora lo ritenesse necessario, marcare un avviso come "visualizzato" nascondendolo.

2.2 Requisiti non funzionali

Le modalità secondo le quali saranno offerte le funzionalità sopracitate sono le seguenti:

- **Usabilità** L'applicazione è stata sottoposta a una ricerca nel dettaglio di "User Personas".
- **Scalabilità** Il sistema deve avere un back-end in cloud scalabile per adattarsi a frequenze di accesso elevate.
- **Password policy security** Le password dell'utente verrano salvate in cloud con password crittografata.
- **Utilizzo di single-activity, multi-fragment** L'applicazione utilizza per fluidità e facile gestione il pattern di single-activity e multi-fragment in modo da dare precisi scopi alle activity che gestiscono fragment comuni.

2.3 Requisiti di dominio

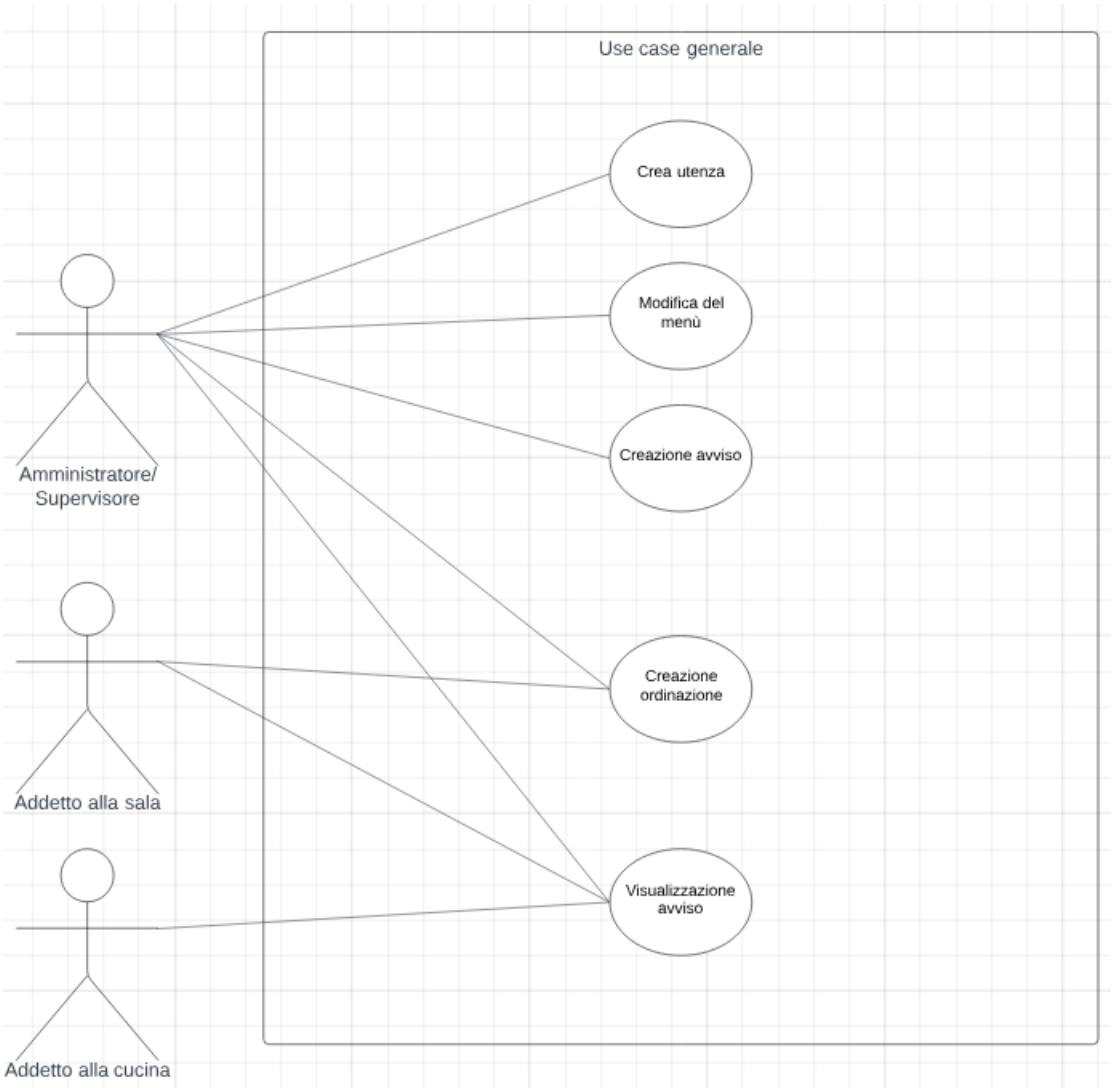
- **ISO/IEC** Il sistema deve essere conforme alle direttive *ISO/IEC* del trattamento dei dati privati su servizi di hosting in cloud.
- **GDPR** Il sistema segue le direttive euorpee sulla *GDPR* e quelle delle policy della privacy.

2.4 Modellazione dei casi d'uso

Di seguito sono descritti in dettaglio gli use-case scelti dal team di sviluppatori, al fine di dare un'idea concreta e chiara sul funzionamento del software sviluppato.

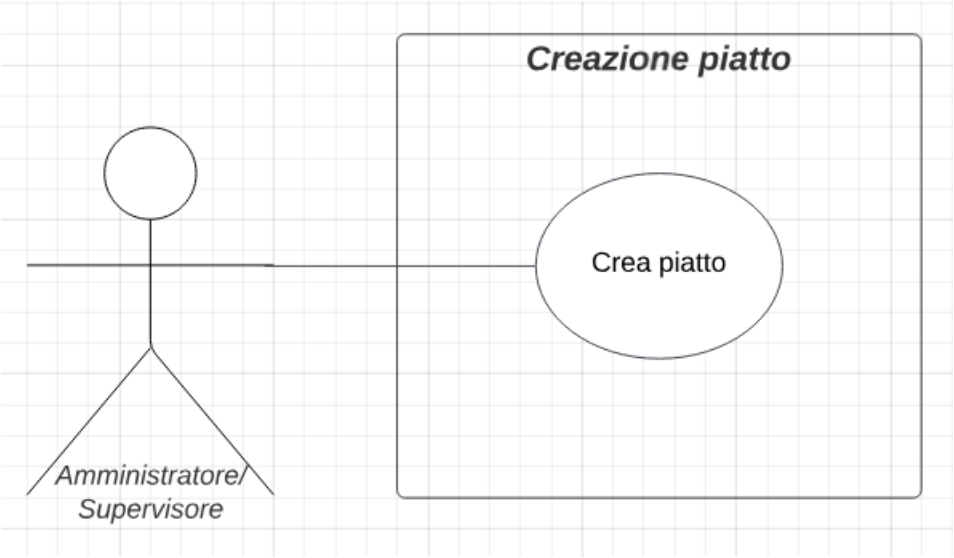
2.4.1 Use-case generale

Questo use-case è nato dopo aver attentamente valutato le richieste da parte degli stakeholders, infatti tale use-case contiene tutte le funzionalità richieste, mostrate in maniera semplificate solo ed esclusivamente per dare un idea generale delle possibilità offerte dal software.



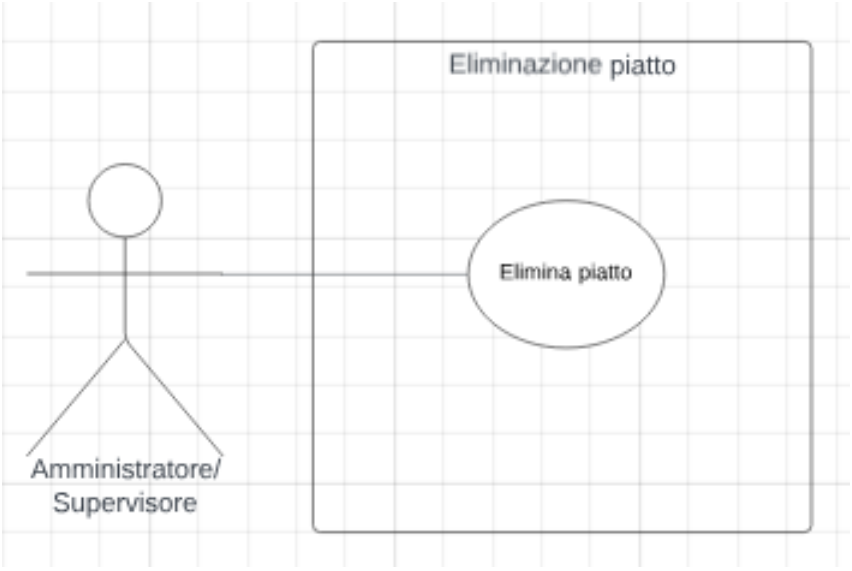
2.4.2 Creazione piatto

La creazione del piatto è una delle funzionalità più elaborate, in quanto, non basterà essere un amministratore/supervisore e inserire i dati (nome, descrizione, allergeni, categoria e prezzo) necessari alla creazione di un piatto. Verrà infatti controllata l'esistenza del piatto e, nel caso di esito positivo il piatto non verrà creato, altrimenti verrà aggiunto alla categoria selezionata.



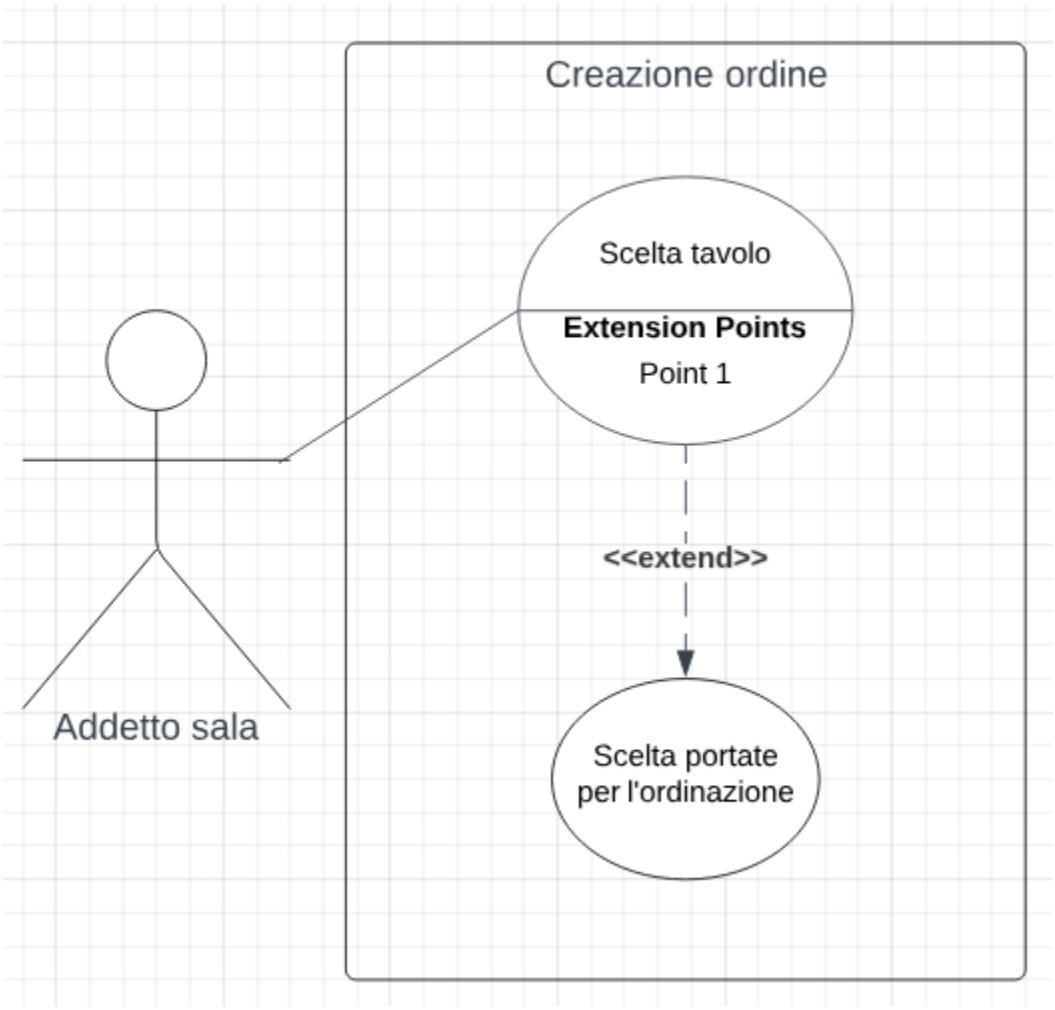
2.4.3 Eliminazione piatto

L'eliminazione del piatto è tutto sommato una funzionalità semplice ed autoesplicativa. Viene semplicemente scelto, da un amministratore/supervisore, il piatto che si desidera eliminare, una volta eliminato ci sarà una richiesta di conferma di eliminazione.



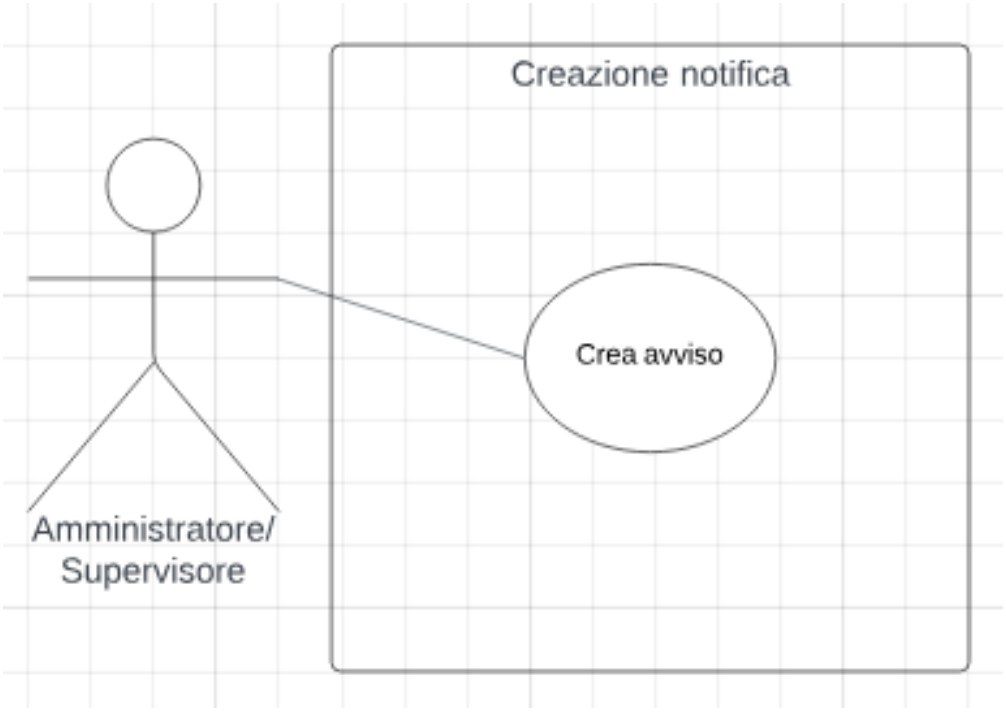
2.4.4 Creazione ordinazione

La creazione di un ordinazione è un requistio fondamentale per la corretta gestione di un'attività di ristorazione che sia essa una tavola calda, un Autogrill o un classico risotrante. Per prima cosa è necessario che l'utente registrato che prova a creare un ordinazione è un "addetto alla sala". Infatti nel caso in cui l'utente registrato non appartiene a questa categoria, non gli sarà permesso a priori la creazione di un ordinazione. Per creare un ordinazione è necessario scegliere un tavolo (tramite l'identificativo associato), e selezionare i piatti che si vogliono aggiungere all'ordinazione (sarà possibile registrare più ordinazioni per lo stesso tavolo in momenti separati).



2.4.5 Creazione notifica

La creazione di un avviso (notifica), è un aspetto fondamentale del software in quanto permette all'amministratore/supervisore di comunicare informazioni importanti indistintamente ad ogni utente. Infatti gli unici utenti con permessi per creare un avviso (notifica) sono proprio l'amministratore e i supervisori.



2.5 Tabelle cockburn

Ci è stato richiesto di presentare, tra i casi d'uso assegnati, quattro casi specifici a scelta descrivendoli tramite **tabelle di Cockburn**.

Cosa sono? Le tabelle di Cockburn (create da *Alistair Cockburn*, dal quale prendono il nome) sono un formalismo di rappresentazione dei casi d'uso.
Sono la rappresentazione di un main scenario (nel nostro primo caso la creazione di un'ordinazione) nel quale uno o più attori interagiscono tra loro attraverso l'invocazione di trigger e descrivendo (in formato tabellare) gli eventi.

Per garantire una certa coerenza al committente e una comprensione a tutto tondo abbiamo scelto gli stessi casi esplorati nella presentazione dei casi d'uso.

2.5.1 Creazione ordinazione

Use Case #1	Creazione ordinazione		
Goal in Context	Si vuole creare una nuova ordinazione		
Preconditions	L'utente deve essere autenticato correttamente ed in **M2**		
Success End Conditions	L'ordinazione è stata creata		
Failed End Conditions	L'ordinazione non è stata creata		
Primary Actor	Utente autenticato		
Trigger	L'utente autenticato preme il tasto "crea ordinazione"		
Description	Step	Utente autenticato	Sistema
	1	Preme "Aggiungi ordinazione"	
	2		Mostra M3
	3	Seleziona i piatti da inserire	
	4	Conferma l'ordine	
	5		Chiude M3
	6		Mostra messaggio "ordinazione creata"
Extensions#1: Categoria o piatto non validi	Step	Utente autenticato	Sistema
	6.1		Mostra messaggio "categoria o piatti non validi"
Extensions#2: Impossibile connettersi	Step	Utente autenticato	Sistema
	6.2		Mostra messaggio "Errore di connessione"

2.5.2 Eliminazione piatto

Use Case #2	Eliminazione piatto		
Goal in Context	L'utente vuole eliminare un piatto dal menù		
Preconditions	L'utente deve essere autenticato correttamente ed in **M4**		
Success End Conditions	Il piatto è stato eliminato		
Failed End Conditions	Il piatto non è stato eliminato		
Primary Actor	Utente autenticato		
Trigger	L'utente autenticato preme il tasto "elimina" sul piatto desiderato		
Description	Step	Utente autenticato	Sistema
	1		Mostra M5
	2	Clicca "ELIMINA"	
	3		chiude M5
	4		Mostra messaggio "piatto eliminato"
Extensions#1: L'utente clicca "INDIETRO"	Step	Utente autenticato	Sistema
	2.1	Clicca "INDIETRO"	
	3.1		Chiude M5
Extensions#2: Impossibile connettersi	Step	Utente autenticato	Sistema
	4.2		Mostra messaggio "Errore di connessione"

2.5.3 Creazione piatto

Use Case #3	Creazione piatto		
Goal in Context	L'utente vuole creare un piatto		
Preconditions	L'utente deve essere autenticato correttamente ed in **M6**		
Success End Conditions	Il piatto è stato creato		
Failed End Conditions	Il piatto non è stato creato		
Primary Actor	Utente autenticato		
Trigger	L'utente autenticato preme il tasto "crea piatto" in **M6**		
Description	Step	Utente autenticato	Sistema
	1	Clicca "crea piatto"	
	2		chiude M7
	3	Riempie i campi	
	4	Conferma la creazione	
	5		Chiude M7
	6		Mostra messaggio "piatto creato"
Extensions#1: campi non validi	Step	Utente autenticato	Sistema
	5.1		Svuota i campi inseriti
	6.1		Mostra messaggio "i campi inseriti non sono validi
Extensions#2: Uno o più campi vuoti	Step	Utente autenticato	Sistema
	5.2		Svuota i campi inseriti
	6.2		Mostra messaggio "compila tutti i campi"
Extensions#3: Impossibile connettersi	Step	Utente autenticato	Sistema
	5.3		Mostra messaggio "Errore di connessione"

2.5.4 Creazione notifica

Use Case #4	Creazione notifica		
Goal in Context	L'utente vuole creare una notifica		
Preconditions	L'utente deve essere autenticato correttamente ed in **M8**		
Success End Conditions	La notifica è stata creata		
Failed End Conditions	La notifica non è stata creata		
Primary Actor	Utente autenticato		
Trigger	L'utente autenticato preme il tasto "crea notifica" in **M8**		
Description	Step	Utente autenticato	Sistema
	1	Clicca "crea notifica"	
	2		chiude M9
	3	Riempie i campi	
	4	Conferma la creazione	
	5		Chiude M9
	6		Mostra messaggio "notifica creata"
Extensions#1: Uno o più campi vuoti	Step	Utente autenticato	Sistema
	5.2		Svuota i campi inseriti
	5.1		Mostra messaggio "compila tutti i campi"
Extensions#2: Impossibile connettersi	Step	Utente autenticato	Sistema
	5.2		Mostra messaggio "Errore di connessione"

2.6 Mock-Up

Un mock-up , è una realizzazione a scopo illustrativo o meramente espositivo, di un oggetto o un sistema, senza le complete funzioni dell'originale; un mockup può rappresentare la totalità o solo una parte dell'originale di riferimento (già esistente o in fase di progetto), essere in scala reale oppure variata.

La versione del mock-up qui sopra rappresenta le schermate principali dell'applicazione. Nella realizzazione di quest'applicazione c'è stato uno studio del colore e dell'usabilità, con una palette creata appositamente per non stancare gli occhi e rendere il tutto più "vicino" al cibo possibile.
Di seguito andremo a visualizzare i mock-up dei metodi scelti dal team.

2.6.1 Creazione ordinazione

Questa schermata rappresenta l'elenco dei tavoli già presenti nel database, quando verrà selezionato un tavolo premendoci sopra, il sistema aprirà **M2**.

M1: pagina dei tavoli

Una volta aperta **M2** ci verrà presentato l'elenco delle portate del tavolo selezionato (nel nostro caso, il tavolo 1). Quando l'utente cliccherà il pulsante "Aggiungi ordinazione" il sistema aprirà **M3**.

M2: Contenuto del tavolo n°1.

Ora che ci troviamo in **M3** potremo usare i due spinner (categoria e piatto) per selezionare il piatto che desideriamo aggiungere all'ordinazione e quando saremo soddisfatti potremo confermarlo.

M3: Nuova ordinazione per il tavolo n°1.

2.6.2 Eliminazione piatto

La schermata riportata qui sotto mostra il menù del ristorante con la possibilità di eliminare i piatti da esso. Qualora l'utente dovesse cliccare "ELIMINA" su uno dei piatti si aprirà **M5**.

M4: Menù ristorante con piatti da eliminare.

In questa schermata l'utente potrà confermare l'eliminazione del piatto oppure tornare indietro, annullando così l'eliminazione.

M5: Conferma eliminazione piatto.

2.6.3 Creazione piatto

La creazione del piatto è una delle funzionalità più importanti, infatti è quella che ci permette di popolare il menù con piatti creati da noi. Per creare un piatto, bisognerà che l'utente di trovi nella schermata del menù (**M6**).

M6: Schermata del menù.

In questa scermata, tramite i tre puntini in alto a destra, possiamo attivare la funzione di creazione di un piatto, che ci aprirà **M7**.

M7: Schermata del menù.

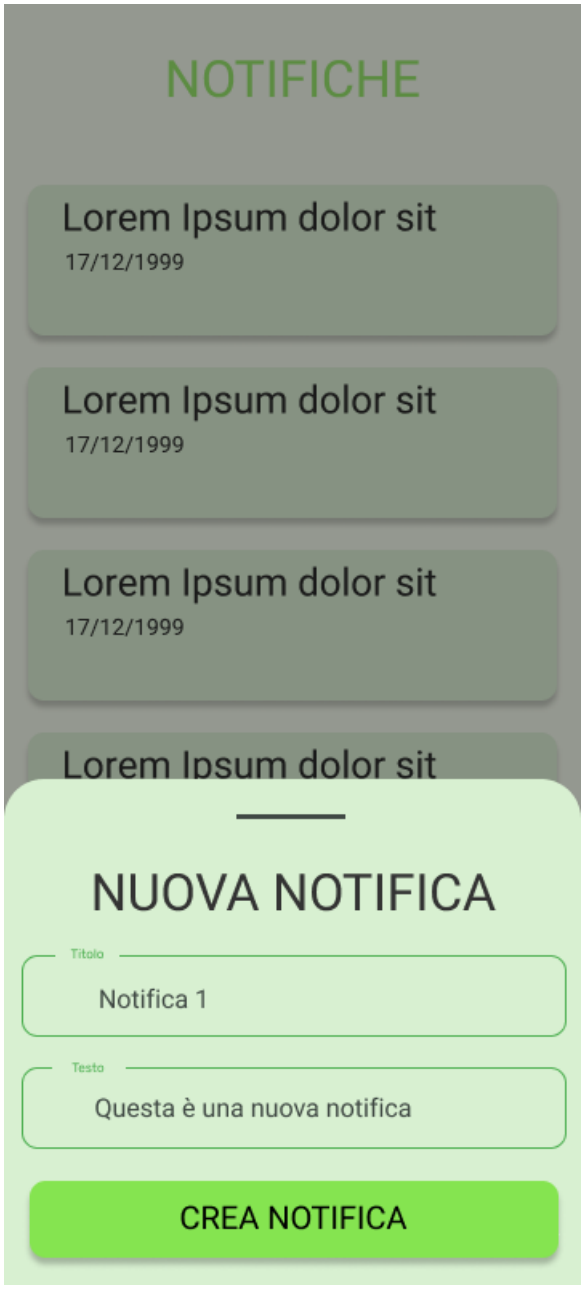
Una volta in questa schermata non bisognerà fare altro che riempire i campi necessari e confermare la craezione.

2.6.4 Creazione notifica

La creazione di una notifica, permette ad un amministratore/supervisore, di mandare un messaggio a tutti i suoi dipendente, così da poter comunicare avvisi importanti indistintamente ad ogni dipendente qualora si voglia. Per farlo, l'utente deve trovarsi in **M8** (↓) e premere "CREA NOTIFICA".

M8: Schermata delle notifiche.

Una volta premuto "CREA NOTIFICA", il sistema mostrerà **M9** (↓), al cui interno ci saranno due campi richiesti (titolo e testo, la data verrà prelevata in automatico dal sistema), una volta compilati, l'utente può premere "CREA NOTIFICA".



M9: Schermata per la creazione di una notifica.

2.7 Individuazione del target a priori

Ratatouille23™ nasce con l'idea di creare un sistema efficiente e affidabile per la gestione di attività di ristorazione. Nello specifico poter permettere ad un responsabile di poter dividere dettagliatamente le mansioni di un dipendente con la possibilità di assumere dei collaboratori (e.g. supervisore). Il target che si può definire da una prima analisi delle funzionalità e casi d'uso, è il seguente:

- Camerieri e manager di ristorante.
- Cuochi, sous chef e aiuto cuoco.
- Proprietario e dirigente di ristorante.

Questa prima analisi può fornirci una visione sommaria delle personas.

2.7.1 Personas

Il focus principale per noi è stato quello di analizzare 2 aspetti che abbiamo ritenuto fondamentali per individuare le user personas così da poter sviluppare un software cucito su misura:

1. **Età:** la fascia d'età che abbiamo potuto notare essere più conforme all'utente ideale è quella dei 18-65 anni, in modo da comprendere uno studente ma anche un cuoco di fama mondiale con alle spalle numerosi anni di esperienza.
2. **Background:** per background s'intende la formazione dell'utente. Infatti un nostro utente può essere uno studente, che lavora il weekend per poter pagare l'affitto da fuorisede, così come può essere un ragazzo con formazione *sommelier* o una ragazza che ha conseguito un corso all'$ALMA$.

Questo ci permette di pianificare sviluppi futuri del nostro software. Potremmo infatti creare una sezione dedicata ai vini, facendo in modo che il sommelier consigli in base ai piatti ordinati, quali vini si accostano meglio.
Potremmo aggiungere una sezione con dei percorsi culinari, studiati in precedenza da un personale esperto e preparato. Così facendo potremo fidelizzare i nostri utenti dandogli completa libertà per la gestione della loro attività.

Lucia Zhang La prima user persona è stata utilizzata per raccoglire il target di utenti nella fascia compresa tra i 25-35 anni, che hanno un'alta competenza con la tecnologia e che sono già avviati, nel mondo del lavoro.

Fig.1: User persona n.1.

Stefano Iamunno La seconda user persona è stata utilizzata per raccoglire il target di utenti nella fascia d'età compresa tra i 18-25 anni, che hanno un'alta competenza tecnologica ma che si sono appena approcciati al mondo del lavoro.

Fig.2: User persona n.2.

Maria colombo La terza user persona è stata utilizzata per raccoglire il target di utenti nella fascia d'età compresa tra i 35-65 anni, che hanno una bassa competenza tecnologica ma con una vasta esperienza nel mondo del lavoro. Sono utenti che con la loro esperienza lavorativa possono assumere pressochè qualsiasi ruolo all'interno di un'attività di ristorazione.

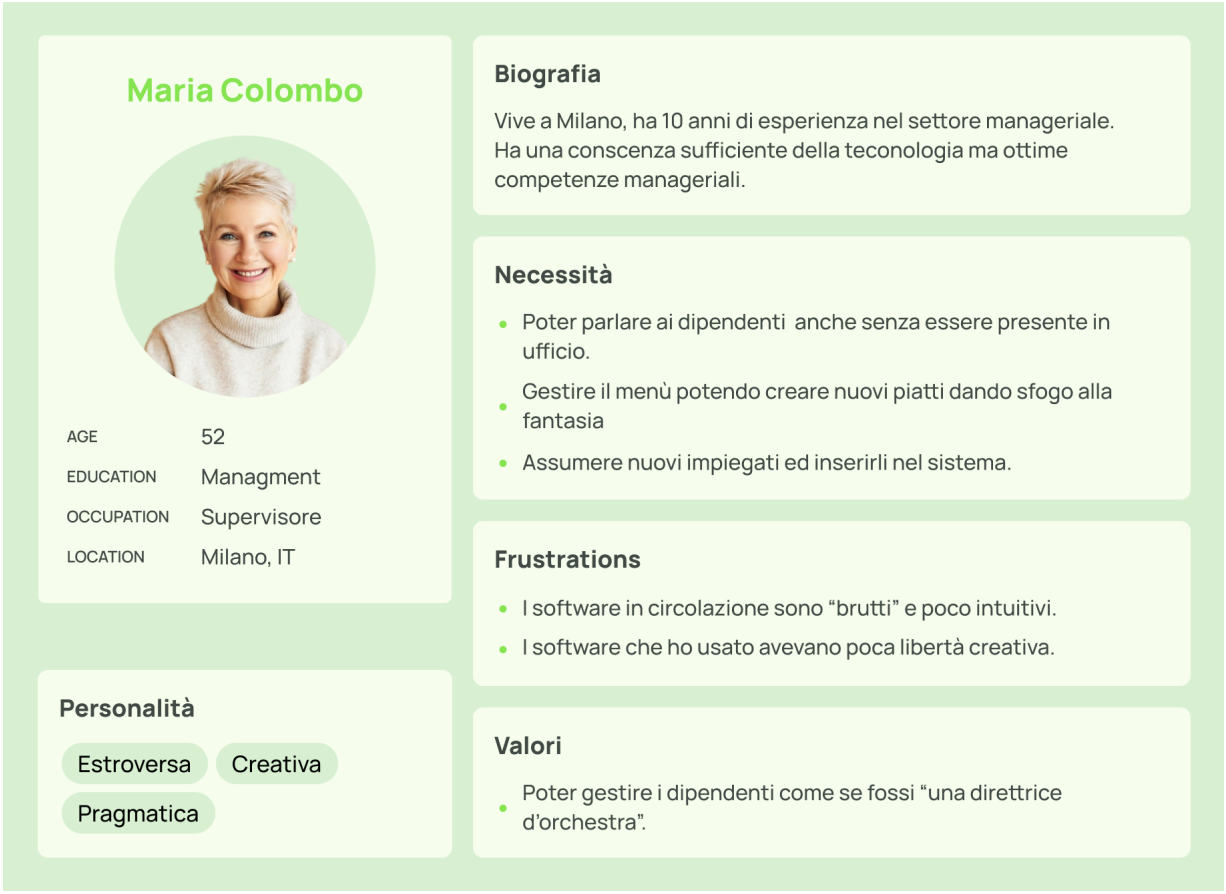


Fig.3: User persona n.3.

2.8 Valutazione dell'usabilità a priori

> L'usabilità di un prodotto è il grado con cui esso può essere usato da specificati utenti per raggiungere specificati obiettivi con efficacia, efficienza e soddisfazione in uno specificato contesto d'uso.

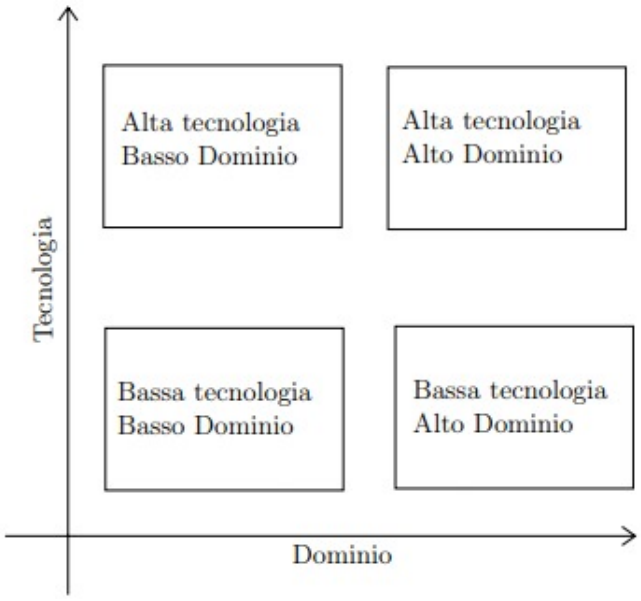
Facendo riferimento allo standard **ISO 9241**(↑), abbiamo basato la valutazione dell'usabilità a priori principalmente su questa definizione in quanto semplice ed efficace, e sulle modalità a nostra disposizione.
In particolare, per ottenere una valutazione dell'usabilità a priori oggettiva e corretta, abbiamo speso molto tempo nella fase di progettazione, della progettazione UI e delle funzionalità, in modo da poter riprodurre dei prototipi della nostra applicazione quanto più realistici possibile. Questo ci ha permesso di apportare poche ma essenziali modifiche alla versione finale, rendendo il tutto più robusto e affidabile.

Per realizzare i mock-up abbiamo usato Figma, quest'ultimo ci permette di creare dei mockup dinamici, simulando il funzionamento dell'applicazione, senza però tralasciare l'aspetto estetico, potendo così fornire una visione concreta della nostra applicaizone.

Le 8 regole d'oro I mock-up, e quindi l'applicazione poi, sono stati realizzati seguendo alcune delle 8 regole di **Shneiderman** con particolare attezione a 4 delle 8 regole:

- **Riscontri informativi**: abbiamo aggiunto dei *pop-up*, schermate di caricamento e messaggi "*toast*", in modo da guidare l'utente durante tutta l'esperienza di utilizzo. In particolare ad ogni azione dell'utente viene notificata tramite messaggio "*toast*", se l'azione è andata a buon fine oppure c'è stato un errore. Abbiamo utilizzato la schermata di caricamento principalemente durante il login, così che se la richiesta al server dovesse richiedere più tempo, l'utente lo saprà e saprà quando la richiesta è terminata. Per quanto riguarda il *pop-up* è stato utilizzato nell'eliminazione del piatto. Infatti quando si proverà ad eliminare un piatto il sistema mostrerà un *pop-up* di conferma per evitare qualsiasi eliminazione fortuita.
- **Riduzione del carico di memoria a breve termine**: è stato garantito non richiedendo informazioni complesse all'utente. Inoltre l'utente non dovrà preoccuparsi di "portare" informazioni da una schermata all'altra.
- **Usabilità universale**: l'applicazione è stata sviluppata tenendo conto di tutte le fasce d'età infatti è estremamente semplice e intuitiva nel suo utilizzo. Le funzionalità sono state implementate nella maniera più semplice possibile, richiedendo meno informazioni possibili e nel caso in cui ci fosse la necessità di chiedere più informazioni, è stato fatto nel modo più graduale possibile.
- **Coerenza a tutti i costi**: le funzionalità sono state sviluppate in modo "simile", creazione notifiche e creazione piatto ad esempio, sono pressochè la stessa funzionalità, rendendo così l'applicazione coerente e di facile utilizzo.

Dopo la realizzazione dei prototipi, è stato effettuato un test di usabilità. Per garantire un risultato oggettivo, sono stati scelti attentamente dei valutatori che andranno utilizzati sui prototipi. Qui sotto è riportato un grafico che rappresenta 4 tipi di valutatori



Ricordandoci "la regola di **Nielsen**", abbiamo scelto 5 tester per l'applicaizone. I 5 utenti metteranno in evidenza la maggior parte dei problemi di usabilità (circa il 75%) più significativi (aggiungerne di più comporterebbe un aumento dei costi inutile). Gli utenti sono i seguenti:

- **Valutatore**: con bassa conoscenza tecnologica, bassa conoscenza del dominio (Fabiana E.).
- **Valutatori** x2: bassa conoscenza tecnologica, alta conoscenza del dominio (Antonio L., Roberto P.).
- **Valutatore**: con alta conoscenza tecnologica, bassa conoscenza del dominio (Corrado R.).
- **Valutatore**: con alta conoscenza tecnologica, alta conoscenza del dominio (Ciro C.).

È importante che gli utenti scelti (al di fuori del gruppo di progetto, in modo da non avere risultati falsati) abbiano caratteristiche diverse, scelti con cura in modo da rappresentare la più ampia fascia d'utenti (per quanto possibile).

2.8.1 Tecnica utilizzata

Per lo svolgimento dei test abbiamo preso spunto dalla tecnica utilizzata sul libro "facile da usare", questo libro infatti spiega molto bene quali dovrebbero essere le cose a cui pensare durante la progettazione e sviluppo di un'applicazione. La tecnica consiste nel realizzare un prototipo interattivo, grazie al quale un tester può avere un idea di quale sarà il funzionamento finale, nella maniera più fedele possibile. Va detto che in questo prototipo non c'è stata alcuna gestione di errori o casistiche speciali, motivo per cui ad ogni tester è stato affiancato il team di sviluppo, per prendere nota e guidare l'utente in caso di errore.

Compiti assegnati Ai nostri utenti sono stati assegnati i seguenti 4 compiti da svolgere:

- **Compito 1**: Cambio password.
- **Compito 2**: Creazione piatto.
- **Compito 3**: Cancellazione piatto.
- **Compito 4**: Visualizzazione notifica.

Nella seguente tabella è possibile visualizzare i risultati dei test:

	COMPITO 1	**COMPITO 2**	**COMPITO 3**	**COMPITO 4**
Fabiana E.	S	S	S	P
Antonio L.	S	F	S	S
Roberto P.	S	S	S	S
Corrado R.	S	P	P	S
Ciro C.	S	P	S	S

Table 1: F = fallimento = 0; P = successo parziale = 0.5; S = successo = 1

Avendo cosi un totale di $17/20 = (15 + (4 * 0.5))$ compiti eseguiti per una percentuale di 85%.

Feedback degli utenti i nostri tester ci hanno evidenziato come:

- L'inserimento del prezzo era lento e incline ad errori, abbiamo sostituito così l'inserimento a mo' di orario con un semplice EditText.
- La visualizzazione delle notifiche era controintuitiva a primo impatto, infatti non tutti sono riusciti subito a trovare il tasto "visualizza", adesso con un semplice tocco della notifica appare il tasto "visualizza".

P.S. in principio erano stati assegnati dei punteggi anche al tempo impiegato, abbiamo però notato che il nostro gruppo di tester si aggirava più o meno sullo stesso tempo impiegato, abbiamo quindi deciso fosse inutile (nel nostro caso) lasciare quei punteggi.

3 Specifica dei casi d'uso

In questa sezione si analizzeranno le classi e come sono relazionate tra loro, daremo un'occhiata ai sequence diagram (in particolare quelli riguardanti due funzionalità dell'applicazione) e agli statechart relativi all'interfaccia grafica. Per fare ciò utilizzeremo:

1. Class diagram.
2. Sequence diagram.
3. State chart

3.1 Classi, oggetti e relazioni di analisi

In questa sezione utilizzeremo i *class diagram* per farci un idea della relazione tra le classi, prima di tutto diamo uno sguardo alla struttura generale che abbiamo creato in fase di analisi:

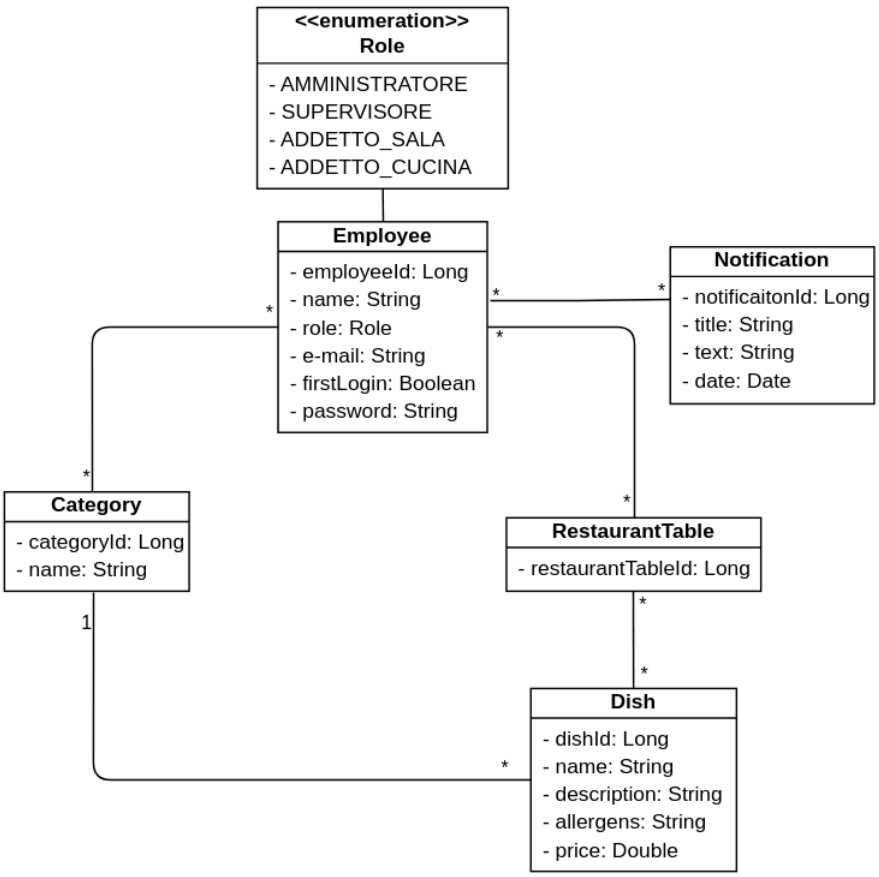


Fig.4: Class diagram delle entità

D'ora in poi invece, i *class diagram* mostrati, saranno quelli delle funzionalità che sono state assegnate al team di sviluppo.

3.2 Class diagram di analisi - Gestione menù

In questo class diagram viene analizzato il punto 3 della traccia ovvero la gestione del menù.

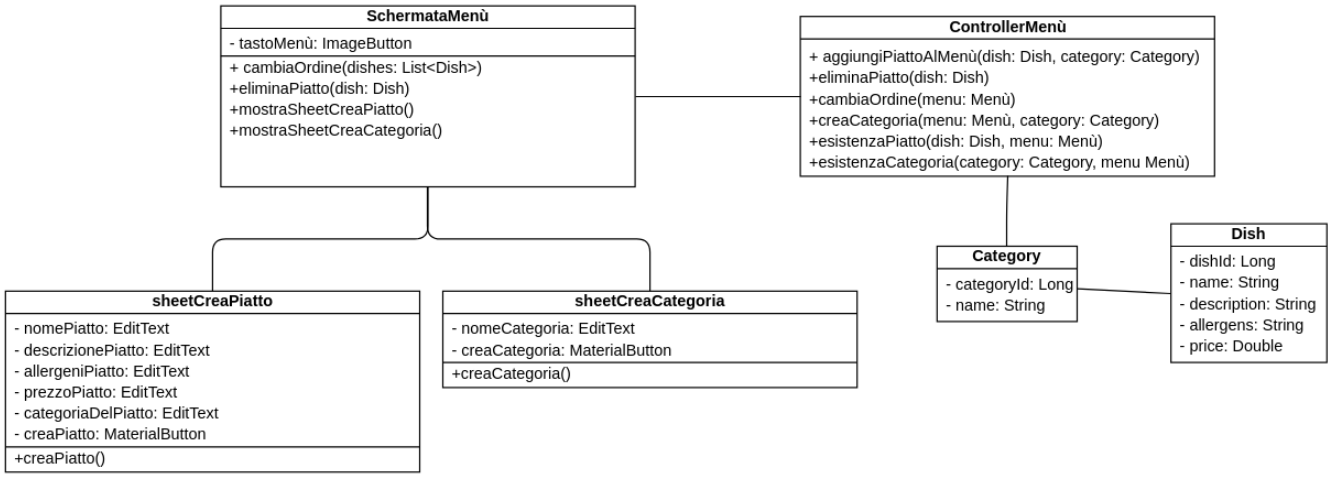


Fig.5: Class diagram della gestione del menù

3.2.1 Class diagram - Creazione utenza

Nel seguente class diagram verrà analizzato il punto 1, quello nel quale si richiede la creazione di un utente e l'eventuale cambio password nel caso in cui si tratti del primo login, il cambio password verrà però affrontato in un *class diagram* successivo, in modo da poter comprendere anche l'accesso alla piattaforma.

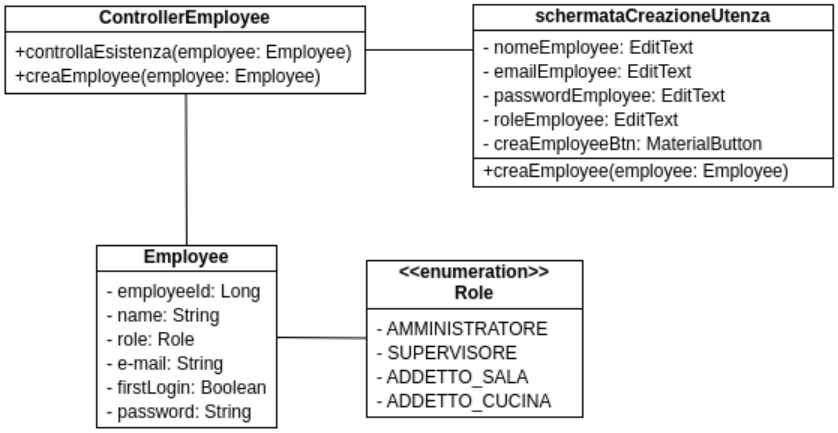


Fig.6: Class diagram per la creazione utenze

3.2.2 Class diagram - Accesso alla piattaforma

Adesso, come detto in precedenza, analizzeremo l'accesso (con conseguente cambio password) alla piattaforma.

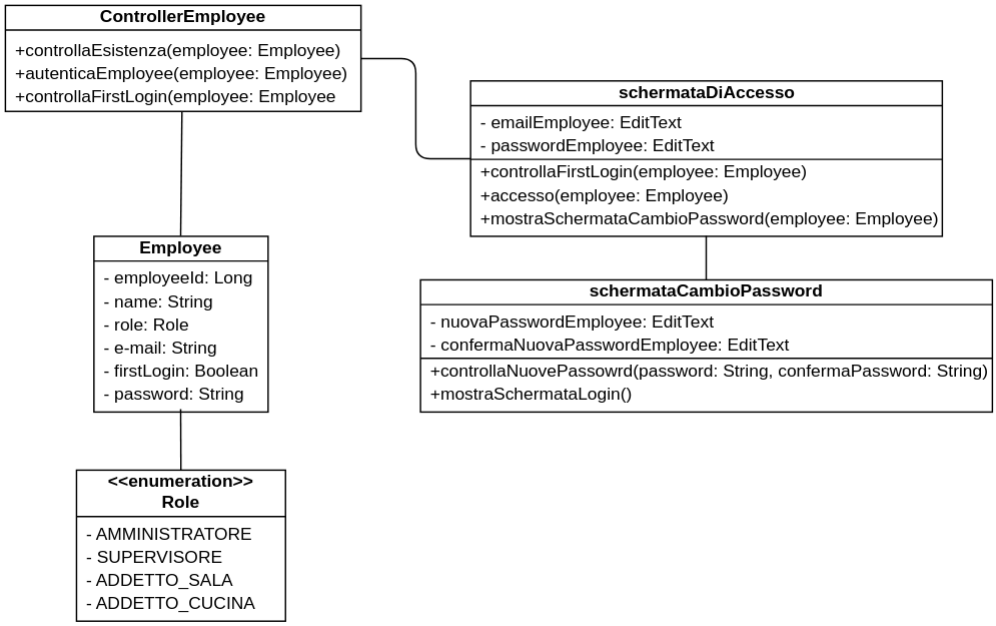


Fig.7: Class diagram dell'accesso alla piattaforma

3.2.3 Class diagram - Gestione tavoli

Il seguente *class diagram* analizza il punto 6, quindi creazione ordinazioni indicando l'identificativo del tavolo e gli elementi del menù da aggiungere al tavolo.

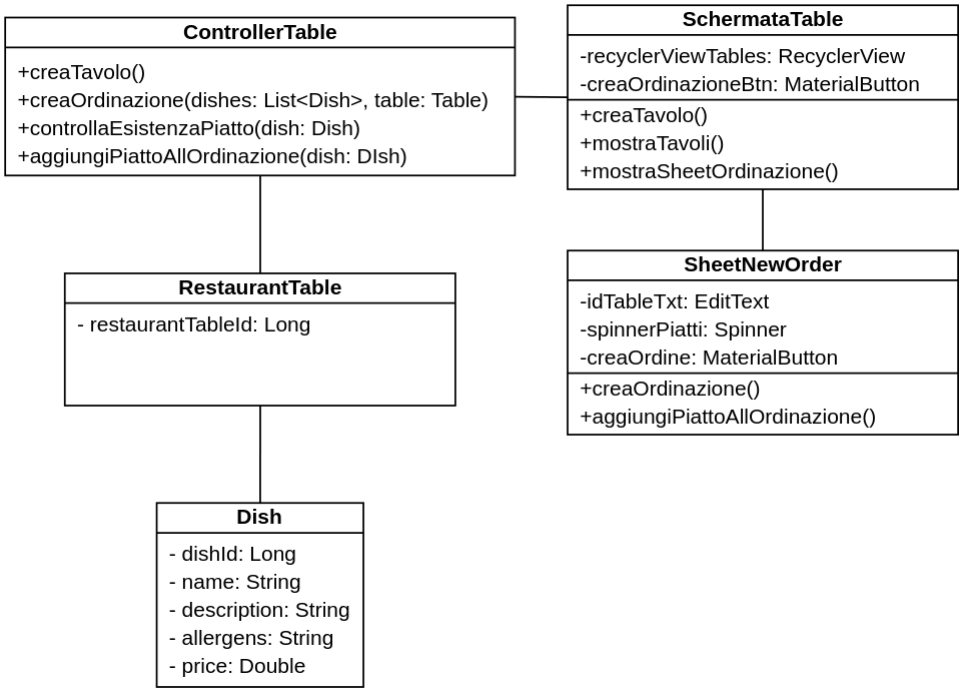


Fig.8: Class diagram della gestione dei tavoli

3.2.4 Class diagram - Gestione notifiche

Il seguente *class diagram* analizza il punto 13, quindi creazione di avvisi (notifiche), al quale verrà aggiunta la visualizzazione delle notifiche.

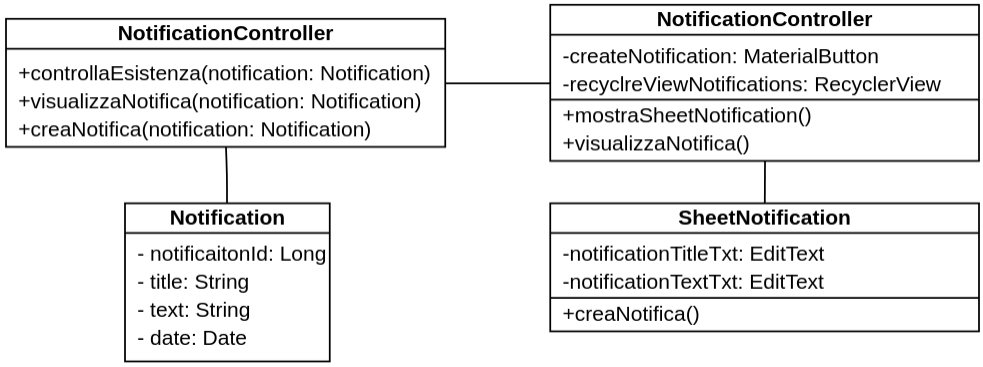


Fig.9: Class diagram della gestione delle notifiche

3.3 Sequence diagram

In questa sezione verranno analizzati due casi d'uso ritenuti fondamentali per questo progetto, in particolare analizzeremo la **creazione di un piatto**, in quanto parte fondamentale per la gestione di un'attività di ristorazione e la **gestione delle notifiche**, quest'ultima è stata scelta poiché tutti i dipendenti possono ricevere e visualizzare notifiche ed è fondamentale ai fini del corretto svolgimento dell'attività.

3.3.1 Sequence diagram - Creazione piatto

In questo *sequence diagram* l'attore è un superivsore o l'amministratore, sono gli unici due ruoli con i permessi per la creazione di un piatto. I controlli che verranno fatti saranno sull'esistenza della categoria selezionata, verrà controllato se il piatto è già presente ed eventualmente se uno o più campi sono vuoti o non validi.

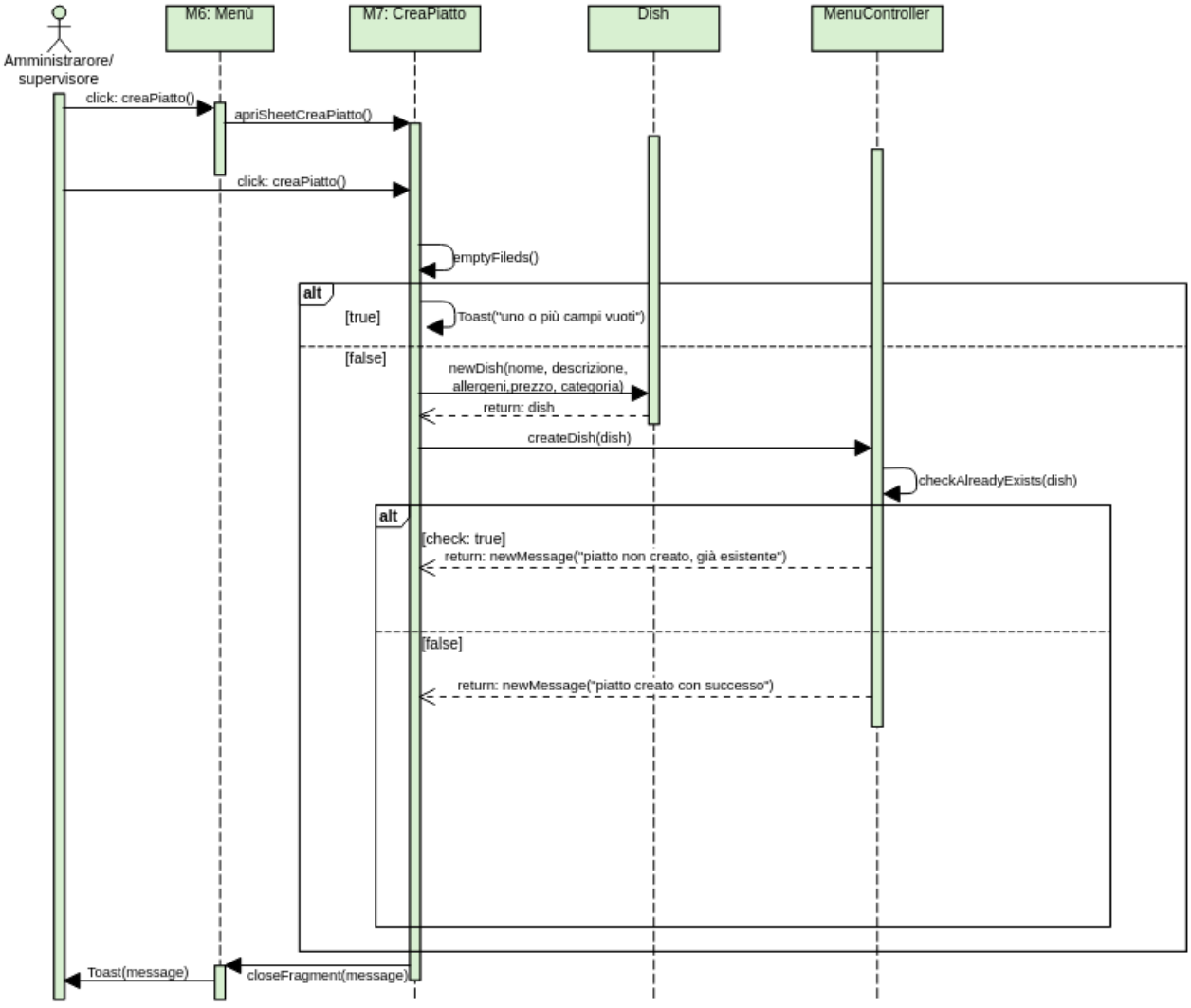


Fig.10: Sequence diagram della creazione di un piatto

3.3.2 Sequence diagram - Creazione piatto

In questo *sequence diagram* l'attore è un superivsore o l'amministratore, sono gli unici due ruoli con i permessi per la creazione di un piatto. I controlli che verranno fatti saranno sull'esistenza della categoria selezionata, verrà controllato se il piatto è già presente ed eventualmente se uno o più campi sono vuoti o non validi.

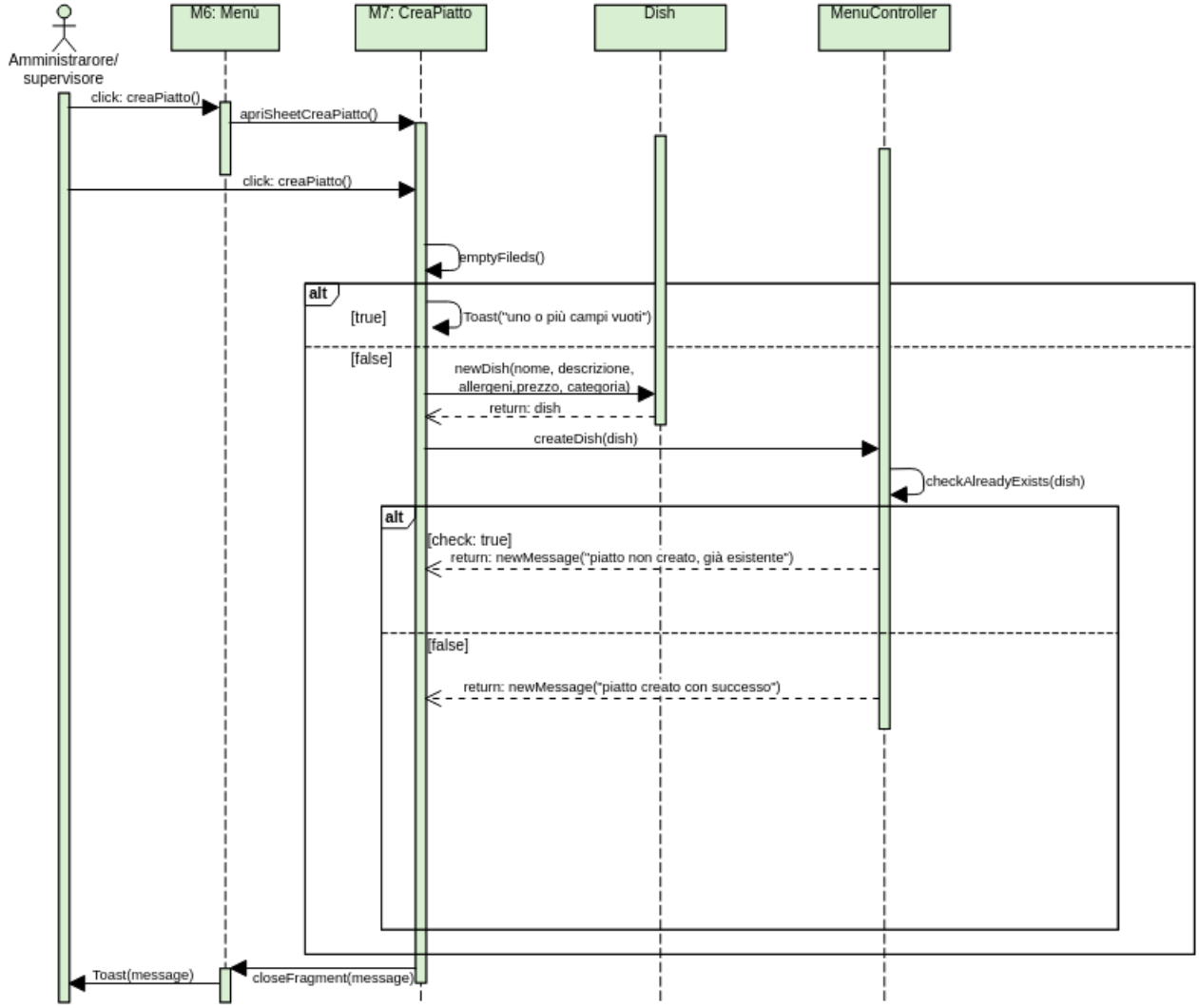


Fig.11: Sequence diagram della creazione di un piatto

3.3.3 Sequence diagram - Creazione notifica

In questo *sequence diagram* l'attore è un superivsore o l'amministratore, sono gli unici due ruoli con i permessi per la creazione di una. I controlli che verranno eseguiti saranno sull'esistenza della notifica, ed eventualmente se uno o più campi sono vuoti o non validi, in entrambi casi se la notifica verrà valutata "non valida", il sistema farà apparire a schermo un messaggio con scritto "errore nella creazione della notifica".

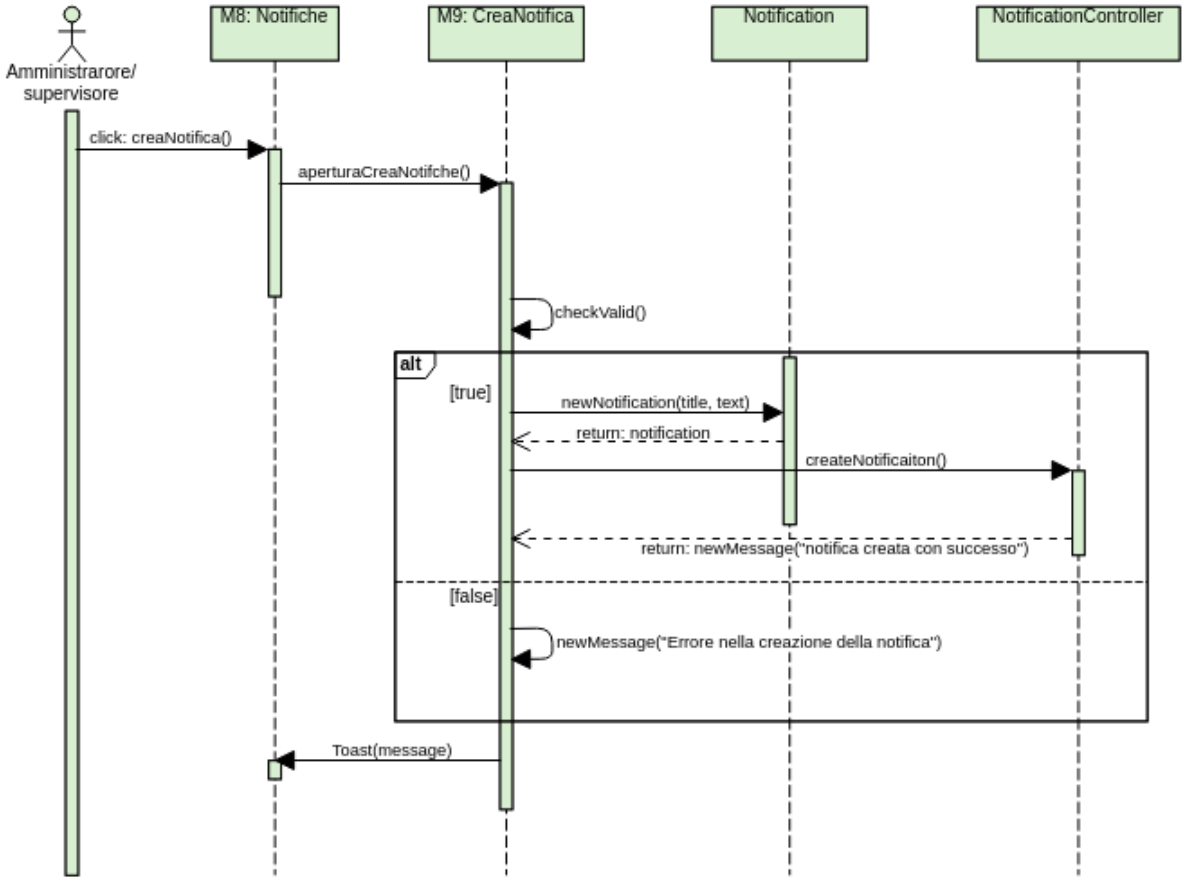


Fig.12: Sequence diagram della creazione di una notifica

3.4 State chart

Gli *state chart* sono un tipo di diagramma che descrivono il comportamento di, tramite eventi che potrebbero accadere per ciascuno stato. Qui di seguito, analizzeremo gli *state chart* dell'interfaccia grafica.

3.4.1 State chart - Notifiche

In questo *state chart* viene analizzata l'interfaccia grafica relativa alla sezione delle notifiche (punto 13).

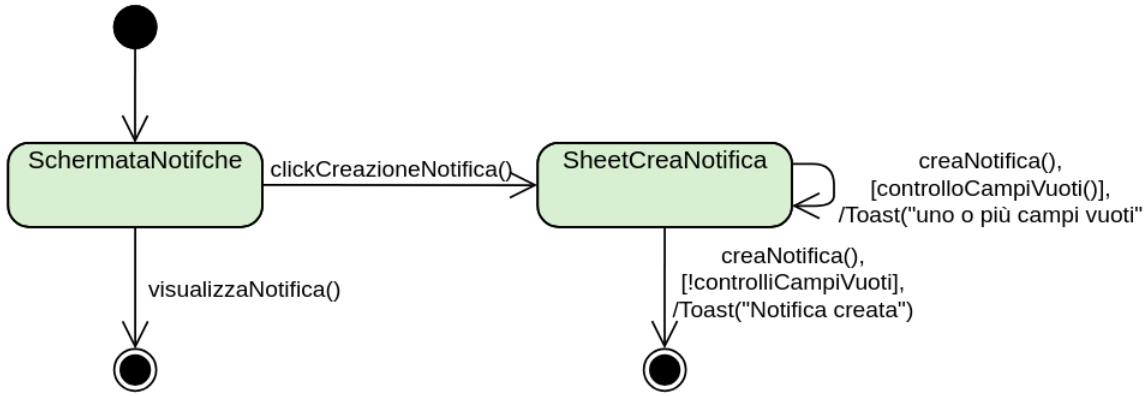


Fig.13: State chart diagram della gestione delle notifiche

3.4.2 State chart - Menù

In questo *state chart* analizzeremo la gestione del menù nel suo insieme, principalmente guardandolo dal lato dell'interfaccia grafica (punto 3).

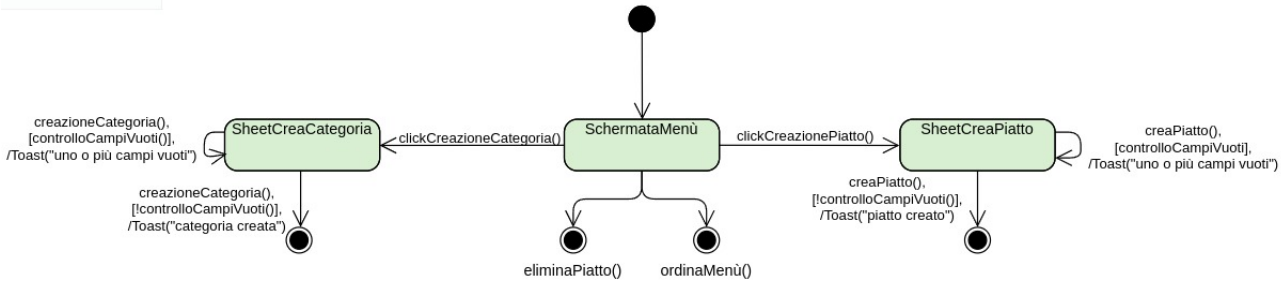


Fig.14: State chart diagram della gestione del menù

3.4.3 State chart - Tavoli

Questo *state chart*, è quello che analizzerà la gestione dei tavoli, sempre dal lato dell'interfaccia grafica (punto 6).

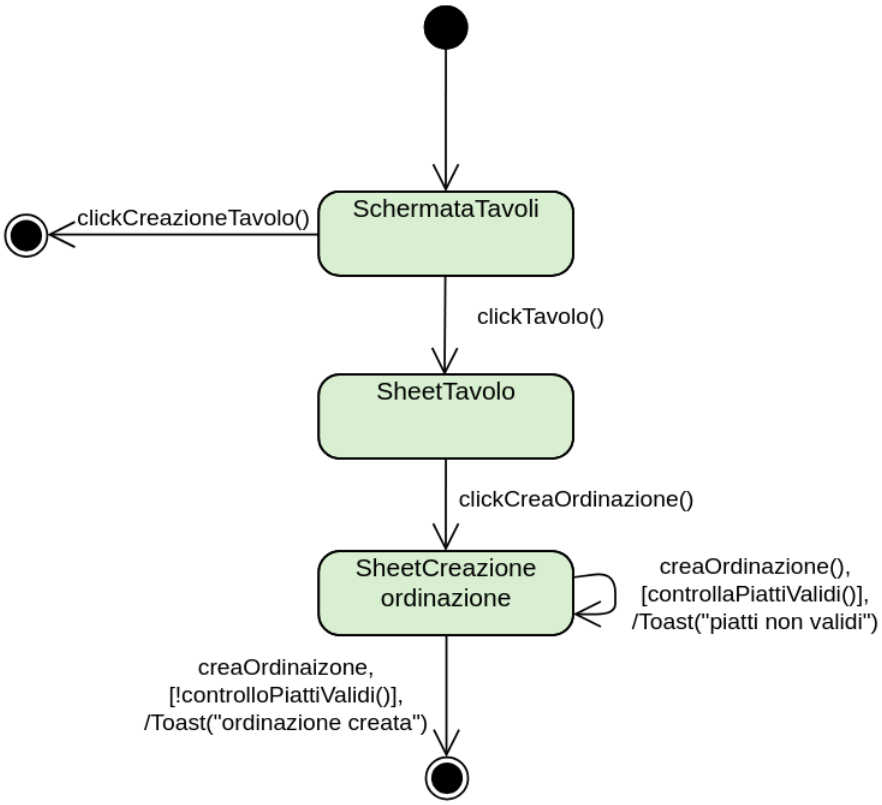


Fig.15: State chart diagram della gestione dei tavoli

3.4.4 State chart - Login e Creazione utenza

Per ultimo, analizzeremo la parte del login e della creazione di un utenza (punto 1).

Creazione utenza - qui sotto è rappresentata la creazione di un utenza.

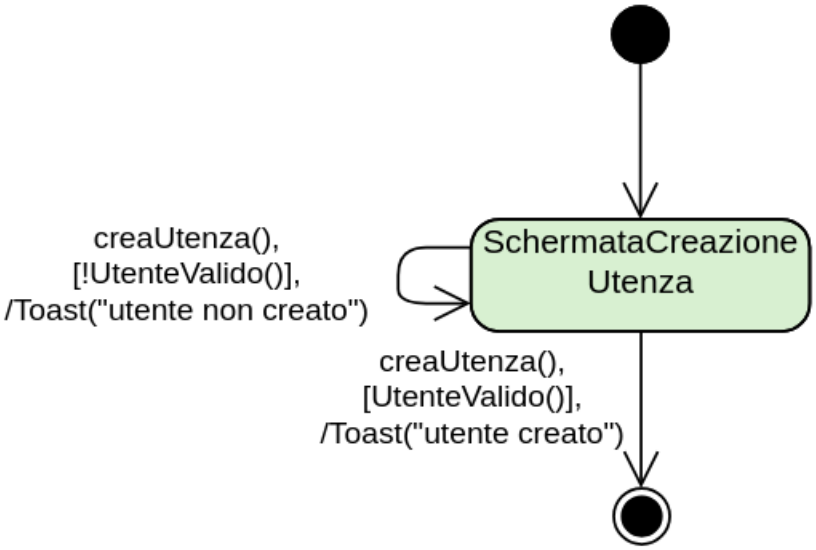


Fig.16: State chart diagram della creazione utenza

Login - Qui sotto invece, è rappresentato il login (punto 1).

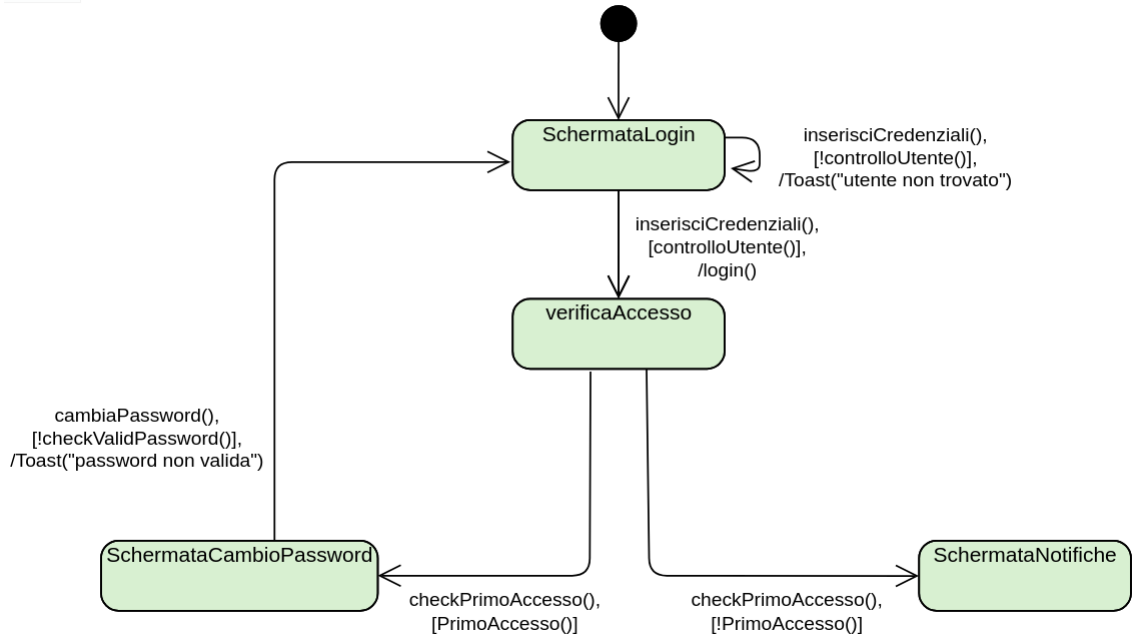


Fig.17: State chart diagram del login

4 Design del sistema

In questa fase viene analizzato il software sviluppato, dando importanza all'architettura del progetto, le tecnologie utilizzate e i motivi per cui le abbiamo utilizzate, i diagrammi delle classi ed i diagrammi di sequenza dei punti scelti dal team.

4.1 Analisi architetturale

Questo progetto è basato sull'architettura **client-server**, è l'architettura più semplice, basata su un server ed un numero arbitrario di client. In quest'architettura i client conoscono tutti i dettagli del server (i servizi), si connettono ad esso e mandando delle richieste (HTTP), sono strettamente dipendenti da esso, infatti, in caso di modifica al server, necessitano di essere ricompilati, mentre il server risponde solo alle richieste che gli arrivano, non curandosi del tipo di client che le invia.

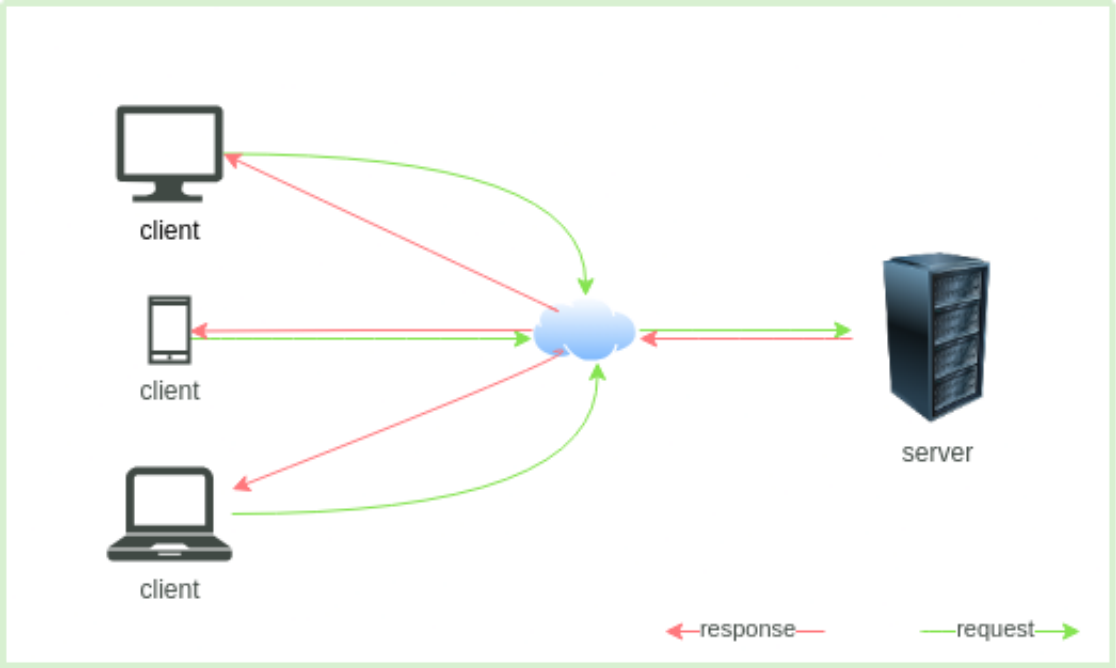


Fig.18: Architettura client-server

Questo garantisce diversi vantaggi:

- **Scalabilità:** Dato che il server non distingue i client richiedenti, siamo in grado, qualora fosse necessario, di scalare il server in modo da accomodare un numero maggiore (o minore) di richieste, permettendoci di risparmiare soldi e risorse.
- **Maggior sicurezza:** Visto che le informazioni critiche, sono archiviate solo sul server, possono essere protette meglio dalle minacce esterne con un maggiore livello di sicurezza.
- **Aggiornamenti** Avendo il server distaccato dai client, è possibile aggiungere nuove funzionalità ad esso, senza dover interrompere le normali operazioni di altri dispositivi.

Architettura cloud Per garantire una fruibilità maggiore ai client, abbiamo deciso di utilizzare tecnologie cloud, questo ci permette di "potenziare" i vantaggi sopracitati, grazie alle tecnologie utilizzate dalla parte cloud.

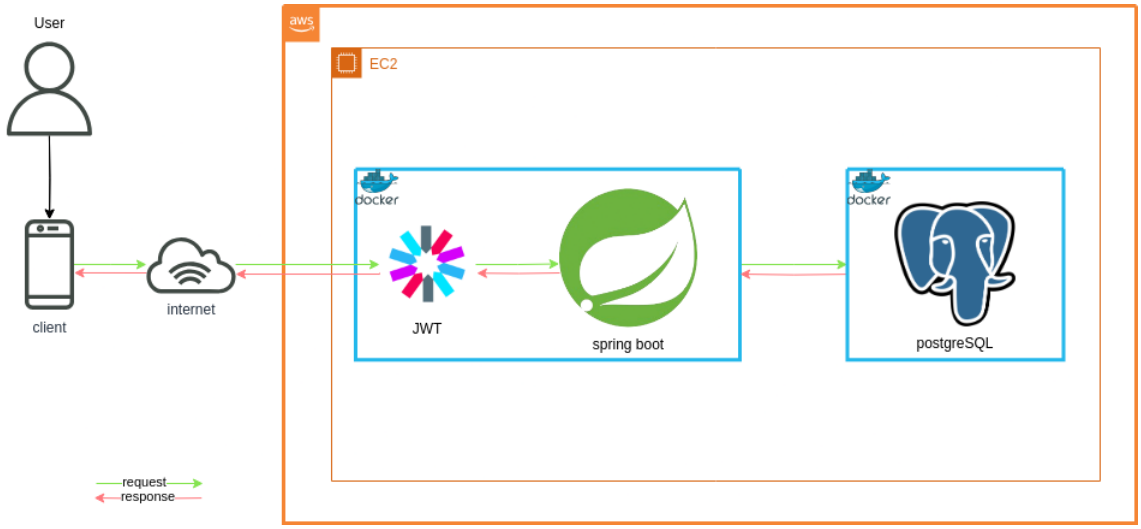


Fig.19: Architettura cloud

Nel nostro caso, abbiamo deciso di suddividiere la parte back-end in docker separati (postgreSQL, springboot), su macchina **EC2** fornita da **AWS**, questo ci permette di avere un sistema nel complesso più modulare e scalabile, garantendo anche la possibilità di avere aggiornamenti futuri, senza stravolgere l'esperienza dell'utente.

4.2 Tecnologie utilizzate

Le tecnologie utilizzate per lo sviluppo di questo progetto sono molteplici, divise in: progettazione, sviluppo, sviluppo della documentazione e testing.

4.2.1 Progettazione

Figma per la progettazione dell'interfaccia grafica, lo sviluppo delle personas e la creazione del brand, ci siamo affidati a figma. Figma è un tool gratuito per la progettazione di interfacce e prototipi, può essere utilizzato sia per la progettazioni di interfacce mobile che web.

Visual paradigm è tool gratuito che permette di creare diversi tipi di diagrammi. È stato utilizzato per lo sviluppo dei class diagram, sequence diagram e state chart.

Tabelle cockburn sono state utilizzate per la specifica dei casi d'uso.

4.2.2 Sviluppo

Springboot è un framework Java, molto affidabile, semplice da implementare e robusto. Utilizzato per lo sviluppo del back-end.

PostgreSQL è un database relazionale. Utilizzato per creare relazioni tra gli oggetti.

Android è il sistema operativo di Google, utilizzato per lo sviluppo del client.

Retrofit è un gestore di chiamate HTTP lato client. Utilizzato per ricevere e mandare richieste HTTP dal client al back-end.

JWT è uno standard, utilizzato per garantire un layer aggiuntivo di sicurezza al nostro progetto.

AWS è un servizio cloud offerto da amazon.

EC2 è un servizio offerto da AWS. Utilizzato per la pubblicazione del back-end in cloud.

Docker è un software che permette di far girare software su ambienti isolabili. Utilizzato per separare il back-end dal front-end.

4.2.3 Sviluppo della documentazione

LaTeX è un mark up language utilizzato per scrivere documenti formali.

draw.io è simile a Visual paradigm. È stato utilizzato dove visual paradigm risultava limitante (e.g. schemi dell'architettura del progetto).

4.2.4 Testing

Mockito è un framework che permette di "mockare" gli oggetti della nostra applicazione, ovvero, permette di simulare degli oggetti (nel nostro caso, sono stati utilizzati per il testing).

JUnit è un framework Java utilizzato per testare la nostra applicazione.

4.3 Class diagram - Desgin

4.4 Sequence diagram - Design

5 Testing

5.1 Testing JUnit

I test sono stati fatti sul back-end, in particolare abbiamo scelto due metodi da fare back box (hideNotification e changePassword) e due white box (createOrder e deleteDish). Per fare ciò sono è stato utilizzato il framework **JUnit** e **Mockito**:

- **JUnit -** è un framework che permette di testare unitariamente applicazioni Java.
- **Mockito -** è un framework Java che permette di simulare oggetti, usato in questo caso per simulare oggetti da passare ai metodi testati.

5.1.1 Black box testing

changePassword vista la mancanza di un recupera password, infatti se si dimentica la password l'unico che può resettarla è un amministratore, è stato scelto in quanto metodo fondamentale. Era necessario che questo metodo non fallisse.

```
public EmployeeResponseDTO changePassword(String password, String email)
```

Per testare questo metodo, sono state analizzate le classi d'equivalenza, in questo caso avendo due **String** come valore non valido si aveva solo **null** mentre gli altri sono tutti validi. Qui sotto riportiamo una tabella (cosi per tutti i metodi) che contiene i valori vali e quelli non validi, sono tutti identificati da una lettera (indica il parametro) e un numero (valido o meno).

Nome parametro	Denominazione parametro	Tipo parametro	Esempio parametro	Validità parametro
password	$A1$	String	null	non valido
password	$A2$	String	"stringa"	valido
email	$B1$	String	null	non valido
email	$B2$	String	"stringa"	valido

Adesso che abbiamo visto il dominio dei parametri, passiamo al testing. Questo metodo è stato testato tramite metodologia **SECT** (Strong Equivalence Class Testing), sono quindi state esplicitate tutte le classi d'equivalenza, arrivando cos' a 4 metodi di testing.

```
@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class ChangePasswordBBT {
    @InjectMocks
    private EmployeeServiceImpl employeeServiceImpl;
    @Mock
    private EmployeeService employeeService;
    @Mock
    private EmployeeRepository employeeRepository;

    @Test
    public void changePassword_A1_B1(){
        assertThrows(UsernameNotFoundException.class, () ->{
        employeeServiceImpl.changePassword(null, null);
    });
    }

    @Test
    public void changePassword_A1_B2(){
```

```
        String email = "paolo";
        assertThrows(UsernameNotFoundException.class, () ->{
            employeeServiceImpl.changePassword(null, email);
        });
    }
    @Test
    public void changePassword_A2_B1(){
        String password= "null";
        assertThrows(UsernameNotFoundException.class, () ->{
            employeeServiceImpl.changePassword(password, null);
        });
    }
    @Test
    public void changePassword_A2_B2(){
        Employee employee = new Employee("paolo",
                "cammardella",
                "prova",
                Employee.Role.ADDETTO_CUCINA,
                false);

        when(employeeRepository.findByEmail(employee.getEmail())).thenReturn(
            employee);
        EmployeeResponseDTO employeeChangePasswd = employeeServiceImpl.
            changePassword(employee.getPassword(), employee.getEmail());
        assertEquals(true, employeeChangePasswd.getRequestSuccess());
    }
}
```

hideNotification abbiamo deciso di testare questo metodo, in quanto è fondamentale per la corretta gestione dell'attività di ristorazione. Basti pensare che le notifiche sono l'unica cosa comune a tutti i ruoli, se quindi dovesse esserci un problema ne risentirebbe il maggior numero di utenti.

```
public NotificationResponseDTO hideNotification(Notification
    hiddenNotification, Employee creatorEmployee)
```

Come per il precedente metodo, sono state analizzate le classi d'equivalenza, in questo caso però non è stato cosi semplice, infatti i parametri passati a questo metodo sono più complessi per quanto riguarda la loro composizione. Il valore non valido per i campi di entrambi gli oggetti, è **null**, quello valido invece è una stringa generica.

Employee creatoreEmployee				
Nome parametro	**Denominazione parametro**	**Tipo parametro**	**Esempio parametro**	**Validità parametro**
name	*A*1	String	null	non valido
name	*A*2	String	”stringa”	valido

Notification hiddenNotification				
Nome parametro	**Denominazione parametro**	**Tipo parametro**	**Esempio parametro**	**Validità parametro**
title	$B1$	String	null	non valido
title	$B2$	String	”stringa”	valido
text	$C1$	String	null	non valido
text	$C2$	String	”stringa”	valido
date	$D1$	String	null	non valido
date	$D2$	String	”stringa”	valido
creatoreEmail	$E1$	String	null	non valido
creatorEmail	$E2$	String	”stringa”	valido

```
@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class HideNotificationBBT {
    @InjectMocks
    private NotificationServiceImpl notificationServiceImpl;
    @Mock
    private NotificationService notificationService;
    @Mock
    private NotificationRepository notificationRepository;
    @Mock
    private EmployeeRepository employeeRepository;
    @Mock
    private EmployeeServiceImpl employeeServiceImpl;
    @Mock
    private EmployeeService employeeService;

    Employee employee;
    Notification notification;

    @BeforeEach
    public void init() {
        employee = new Employee();
        employee.setNotifications(new ArrayList<>());
        employee.setName("Paolo");
        employee.setEmail("paolo@cammardella.it");
        employee.setPassword("ciao");
        employee.setRole(Employee.Role.ADDETTO_CUCINA);
        employee.setFirstLogin(false);
        employee.setRestaurantTables(new ArrayList<>());
        employee.setCategories(new ArrayList<>());


        notification = new Notification("notifica", "testo notifica", "
            12/03/1999", "paolo@cammardella.it");
        notification.setEmployees(new ArrayList<>());
        employee.getNotifications().add(notification);
        notification.getEmployees().add(employee);
        when(employeeService.findEmployeeByEmail(employee.getEmail())).
            thenReturn(employee);
        when(notificationRepository.findByTitleAndTextAndCreatorEmailAndDate(
            notification.getTitle(), notification.getText(), notification.
            getCreatorEmail(), notification.getDate())).thenReturn(
            notification);
    }


    @Test
    public void testHideNotificationA1_B1_C1_D1_E1() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setText(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
```

```
        });
    }

    @Test
    public void testHideNotificationA1_B1_C1_D1_E2() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setText(null);
        notification.setDate(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B1_C1_D2_E1() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setText(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B1_C1_D2_E2() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setText(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B1_C2_D1_E1() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B1_C2_D1_E2() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setDate(null);
```

```
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B1_C2_D2_E1() {
        employee.setName(null);
        notification.setTitle(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B1_C2_D2_E2() {
        employee.setName(null);
        notification.setTitle(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C1_D1_E1() {
        employee.setName(null);
        notification.setText(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C1_D1_E2() {
        employee.setName(null);
        notification.setText(null);
        notification.setDate(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C1_D2_E1() {
        employee.setName(null);
        notification.setText(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
```

```
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C1_D2_E2() {
        employee.setName(null);
        notification.setText(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C2_D1_E1() {
        employee.setName(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C2_D1_E2() {
        employee.setName(null);
        notification.setText(null);
        notification.setDate(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C2_D2_E1() {
        employee.setName(null);
        notification.setCreatorEmail(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA1_B2_C2_D2_E2() {
        employee.setName(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
```

```
    public void testHideNotificationA2_B1_C1_D1_E1() {
        notification.setTitle(null);
        notification.setText(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);
        employee.getNotifications().add(notification);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C1_D1_E2() {
        notification.setTitle(null);
        notification.setText(null);
        notification.setDate(null);

        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C1_D2_E1() {
        notification.setTitle(null);
        notification.setText(null);
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C1_D2_E2() {
        notification.setTitle(null);
        notification.setText(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C2_D1_E1() {
        notification.setTitle(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C2_D1_E2() {
        notification.setTitle(null);
        notification.setDate(null);
        assertThrows(RuntimeException.class, () -> {
```

```
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C2_D2_E1() {
        notification.setTitle(null);
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B1_C2_D2_E2() {
        notification.setTitle(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C1_D1_E1() {
        notification.setText(null);
        notification.setDate(null);
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C1_D1_E2() {
        notification.setText(null);
        notification.setDate(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C1_D2_E1() {
        notification.setText(null);
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C1_D2_E2() {
        notification.setText(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }
```

```
    @Test
    public void testHideNotificationA2_B2_C2_D1_E1() {
        notification.setDate(null);
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C2_D1_E2() {
        notification.setText(null);
        notification.setDate(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C2_D2_E1() {
        notification.setCreatorEmail(null);
        assertThrows(RuntimeException.class, () -> {
            notificationServiceImpl.hideNotification(notification, employee);
        });
    }

    @Test
    public void testHideNotificationA2_B2_C2_D2_E2() {
        NotificationResponseDTO response = notificationServiceImpl.
            hideNotification(notification, employee);
        assertEquals(true, response.getIsSuccesful());
    }
}
```

5.2 White box testing

Per il testing *white box* sono stati scelti due metodi più semplici da testare, fondamentale si ma meno importanti dei due fatti tramite *black box testing*. Questo perché i metodi che vedremo di seguito devono essere si affidabili, ma un errore nell'esecuzione di uno di questi due metodi, sarà meno impattante sull'esperienza dell'utente.

createOrder il metodo per la creazione di un ordinazione permette ad un "addetto alla sala" di aggiungere ordinazioni ad un tavolo. Di seguito viene riportato il metodo per intero:

```
    @Override
    public RestaurantTableDTO createOrder(RestaurantTable restaurantTable,
        Dish dish) {
        if (restaurantTable != null && dish != null) {
            Dish newOrderDish = dishRepository.findByName(dish.getName());
            newOrderDish.getRestaurantTables().add(restaurantTableService.
                findById(restaurantTable.getId()));
            dishRepository.save(newOrderDish);
            restaurantTable.getDishes().add(newOrderDish);
            return new RestaurantTableDTO(restaurantTable,
```

```
                    true, "Piatto: " + dish.getName() + " aggiunto al tavolo:
                        " + restaurantTable.getId().toString());
        }
        return new RestaurantTableDTO(restaurantTable,
                false, "Piatto o tavolo non valido");
    }
```

Osservando questo metodo, possiamo ricavare il *grafo del flusso di controllo*, che ci permetterà di seguito di testare il metodo nel miglior modo possibile:

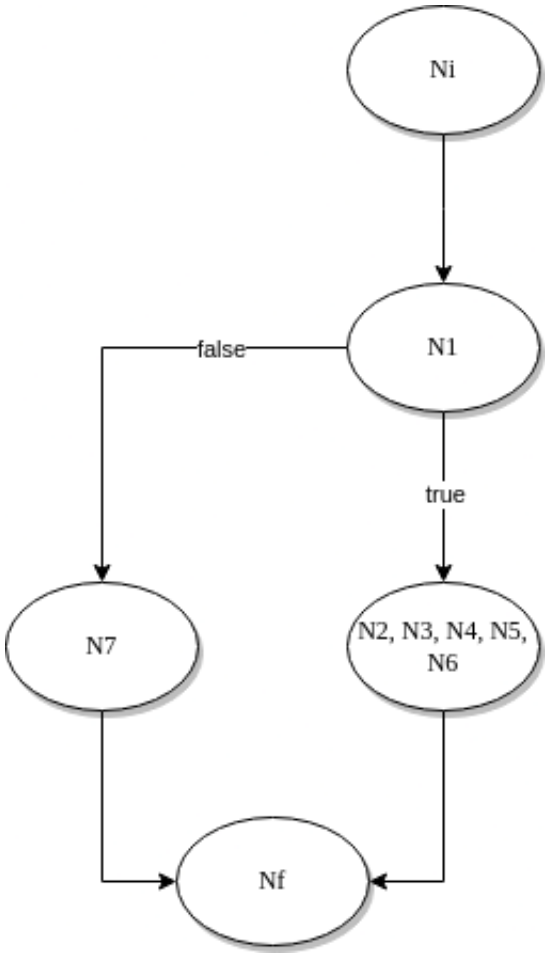


Fig.20: Grafo del flusso di controllo per createOrder

Osservando questo grafo, possiamo ricavare due metodi di testing in modo da ottenere un cosiddetto "**branch coverage**", ovvero facciamo in modo che ogni caso sia coperto da test:

- testCreateOrder1()
- testCreateOrder2()

```
@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class CreateOrderWBT {
```

```
    @InjectMocks
    private DishServiceImpl dishServiceImpl;

    @Mock
    private DishService dishService;
    @Mock
    private DishRepository dishRepository;
    @Mock
    private RestaurantTableService restaurantTableService;

    @Test
    public void testCreateOrder1() {
        RestaurantTable restaurantTable = new RestaurantTable();
        restaurantTable.setId(1);

        Dish dish = new Dish();
        dish.setId(2);
        dish.setName("Pasta");
        dish.setCategory(new Category(1, "PRIMI", new ArrayList<Dish>()));

        when(dishRepository.findByName("Pasta")).thenReturn(dish);
        when(restaurantTableService.findById(1)).thenReturn(restaurantTable);

        RestaurantTableDTO response = dishServiceImpl.createOrder(
            restaurantTable, dish);
        assertEquals(restaurantTable, response.getRestaurantTable());
        assertEquals(true, response.getIsSuccessful());
        assertEquals("Piatto: Pasta aggiunto al tavolo: 1", response.
            getRequestInfo());
    }

    @Test
    public void testCreateOrder2() {
        RestaurantTableDTO response = dishServiceImpl.createOrder(null, null);
        assertEquals(null, response.getRestaurantTable());
        assertEquals(false, response.getIsSuccessful());
        assertEquals("Piatto o tavolo non valido", response.getRequestInfo());

    }
}
```

deleteDish il metodo per l'eliminazione di un piatto, permette ad un amministratore/supervisore di eliminare un piatto dal menù. Abbiamo deciso di testare questo metodo tramite *white box testing* in quanto, si fondamentale, ma un errore in questo metodo, può essere ovviato mandando una notifica con i piatti non disponibili. Ciò non vuol dire che il metodo può non funzionare. Di seguito è riportato il metodo per intero:

```
    @Override
    @Transactional
    public MenuResponseDTO deleteDish(String dishName, Category category) {
        if (dishRepository.existsByName(dishName)) {
            Dish dish = dishRepository.findByName(dishName);
            categoryService.printCategoryByName(category.getName()).
                getCategoryDishDTO().getCategory().getDishes().remove(dish);
            dishRepository.deleteByName(dishName);
            log.info("Piatto eliminato con successo: {}", dish);
```

```
            return new MenuResponseDTO(null, "Piatto eliminato con successo",
                true);
        }
        log.info("Piatto non trovato!");
        return new MenuResponseDTO(new CategoryDishDTO(categoryService.
            printCategoryByName(category.getName()).getCategoryDishDTO().
            getDishes(), category), "Piatto inesistente", false);
    }
```

Da questo metodo, come fatto per il precedente, possiamo ricare un grafo del flusso di controllo:

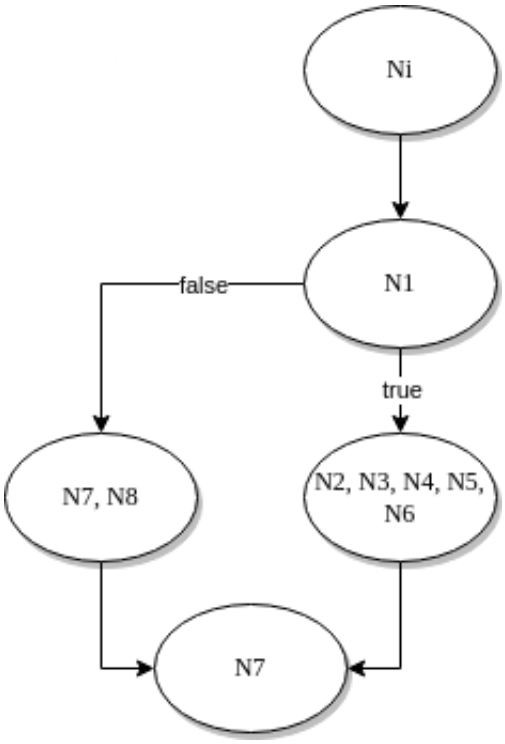


Fig.21: Grafo del flusso di controllo per deleteDish

Come fatto per il precedente metodo, individuiamo i metodi che ci permettono di ottenere il branch coverage:

- testDeleteDish1()
- testDeleteDish2()

Di seguito vengono riportati i metodi di testing.

```
@ExtendWith(MockitoExtension.class)
@ExtendWith(SpringExtension.class)
public class DeleteDishWBT {
    @InjectMocks
    private DishServiceImpl dishServiceImpl;
    @Mock
    private DishService dishService;
    @Mock
    private DishRepository dishRepository;
```

```
@Mock
private CategoryRepository categoryRepository;
@Mock
private CategoryService categoryService;

@Test
public void testDeleteDish1(){
    String dishName = "pollo";

    Category category = new Category();
    category.setName("Primi");
    category.setDishes(new ArrayList<>());

    Dish dish = new Dish();
    dish.setId(2);
    dish.setName(dishName);
    dish.setCategory(category);

    when(dishRepository.existsByName(dishName)).thenReturn(true);
    when(dishRepository.findByName(dishName)).thenReturn(dish);
    when(categoryService.printCategoryByName(category.getName())).
        thenReturn(new MenuResponseDTO(new CategoryDishDTO(category.
        getDishes(), category), "Piatto eliminato con successo", true));

    MenuResponseDTO response = dishServiceImpl.deleteDish(dishName,
        category);
    assertEquals(true, response.getIsSuccessful());
    assertEquals("Piatto eliminato con successo", response.getDetail());
}


@Test
public void testDeleteDish2(){
    String dishName = "pollo";

    Category category = new Category();
    category.setName("Primi");
    category.setDishes(new ArrayList<>());

    Dish dish = new Dish();
    dish.setId(2);
    dish.setName(dishName);
    dish.setCategory(new Category(1, "PRIMI", new ArrayList<Dish>()));


    when(dishRepository.existsByName(dishName)).thenReturn(false);
    when(categoryService.printCategoryByName(category.getName())).
        thenReturn(new MenuResponseDTO(new CategoryDishDTO(category.
        getDishes(), category), "Piatto inesistente", false));

    MenuResponseDTO response = dishServiceImpl.deleteDish(dishName,
        category);
    assertEquals(false, response.getIsSuccessful());
    assertEquals("Piatto inesistente", response.getDetail());
}
```

```
}
```

5.3 Valutazione dell'usabilità sul campo

Per l'usabilità sul campo abbiamo utilizzato alcuni dei metodi già visti per **l'usabilità a priori 2.8**, aggiungendoci però un periodo di testing dell'applicazione in fase finale, da parte dei nostri volontari, che ci hanno garantito dei preziosi feedback ed in fine, abbiamo utilizzato le librerie di logging di Java e di Android, per generare dei costanti log in modo da tenere sotto controllo il corretto funzionamento dell'applicazione.

5.3.1 Compiti assegnati

Come per l'usabilità a priori, abbiamo deciso di assegnare (gli stessi) 4 compiti, ai nostri 5 tester, questa volta però, non siamo più stati noi del team ad inidrizzare gli utenti in base alla risposta data, ma è stato fatto tutto grazie all'applicazione in fase finale. Ricapitolando, abbiamo assegnato 4 compiti:

- **Compito 1**: Cambio password.
- **Compito 2**: Creazione piatto.
- **Compito 3**: Cancellazione piatto.
- **Compito 4**: Visualizzazione notifica.

Raccogliendo i risultati in una tabella:

	COMPITO 1	**COMPITO 2**	**COMPITO 3**	**COMPITO 4**
Fabiana E.	S	S	S	S
Antonio L.	S	P	S	S
Roberto P.	S	S	S	S
Corrado R.	S	S	P	S
Ciro C.	S	S	S	S

Table 2: F = fallimento = 0; P = successo parziale = 0.5; S = successo = 1

Come possiamo notare dalla nostra tabella, non abbiamo più rilevato *fallimenti* (P), e i *successi parziali* (P) sono stati ridotti notevolemente. Abbiamo così raggiunto $(18 + (2 * 0.5)) = 19/20 = 95\%$ di successi. Un netto miglioramento rispetto alla perecentuale precedente (85%).

5.3.2 Feedback degli utenti

Un miglioramento così grande è stato possibile anche grazie ai feedback forniti dai nostri utenti, che ci ha permesso di risolvere le problematiche più comuni e ricorrenti.

5.3.3 Log

Oltre alle metodologie precedenti abbiamo utilizzato **SLF4J** (Simple Logging Facade for Java) per i log lato back-end:

```
log.info("Notifica nascosta con successo: {}", notification);
log.info("Piatto aggiunto: {} al tavolo: {}", newOrderDTO.getDish(), newOrderDTO.getRestaurantTable());
log.info("Piatto creato con successo: {}", dish);
log.info("EmployeeServiceImpl.loginEmployee: Wrong e-mail or password\n{}", employeeResponseDTO);
```

Fig.22: Log tramite SLF4J.

Mentre è stata usata la librerie **Log** di Android, per il log lato front-end.

```
Log.d(TAG, msg: "Ordinazione creata");
Log.d(TAG, msg: "Passaggio alle notifiche, richiesta cambio password avvenuta con successo!");
```

Fig.23: Log tramite *Log*.