



**UNIVERSITÀ  
DEGLI STUDI  
DI BERGAMO**

---

CORSO DI LAUREA MAGISTRALE DI INGEGNERIA INFORMATICA

Dipartimento di Ingegneria Gestionale, dell'Informazione e della  
Produzione



## ServeEasy

Progetto del corso di  
Progettazione, Algoritmi e Computabilità

Prof.ssa  
**Patrizia Scandurra**

Candidati:  
**Giorgio Chirico**  
1068142

**Isaac Maffeis**  
1041473

**Guyphard Ndombasi**    **Salvatore Salamone**  
1092015                1096149

---

Anno accademico 2023-2024



# Indice

<b>Elenco delle figure</b>	<b>3</b>
<b>1 Iterazione 0</b>	<b>7</b>
1.1 Introduzione . . . . .	7
1.2 Requisiti funzionali . . . . .	8
1.2.1 Use case stories: Amministratore . . . . .	8
1.2.2 Use case stories: Cliente . . . . .	9
1.2.3 Use case stories: Cuoco . . . . .	9
1.2.4 Use case stories: Cassiere . . . . .	10
1.2.5 Priorità dei casi d'uso . . . . .	10
1.2.6 Use case diagram . . . . .	11
1.3 Requisiti non funzionali . . . . .	13
1.3.1 Performance . . . . .	13
1.3.2 Integrabilità . . . . .	13
1.3.3 Modificabilità . . . . .	13
1.3.4 Testabilità . . . . .	13
1.3.5 Sicurezza . . . . .	13
1.4 Topologia . . . . .	14
1.5 Toolchain . . . . .	15
1.5.1 Modellazione . . . . .	15
1.5.2 Stack applicativo . . . . .	15
1.5.3 Deployment . . . . .	15
1.5.4 Gestore repository . . . . .	15
1.5.5 Continuous Integration . . . . .	15
1.5.6 Analisi statica . . . . .	15
1.5.7 Analisi dinamica . . . . .	16
1.5.8 Documentazione e organizzazione del team . . . . .	16
1.5.9 Modello di sviluppo . . . . .	16
<b>2 Iterazione 1</b>	<b>17</b>
2.1 Introduzione . . . . .	17
2.2 Use Cases . . . . .	18
2.2.1 Gruppo Sistema . . . . .	18
2.2.2 Gruppo cliente . . . . .	19
2.2.3 Gruppo cuoco . . . . .	19

---

2.3	Component Diagram . . . . .	20
2.3.1	Sistema ServeEasy . . . . .	20
2.3.2	Gestione Comanda . . . . .	21
2.3.3	Gestione Cliente . . . . .	26
2.3.4	Gestione Cucina . . . . .	29
2.4	Database . . . . .	32
2.4.1	Modello Entità-Relazione . . . . .	32
2.4.2	Modello logico . . . . .	33
2.5	Interface Class Diagram . . . . .	35
2.5.1	Interfacce Gestione Comanda . . . . .	35
2.5.2	Interfacce Gestione Cucina . . . . .	36
2.5.3	Interfacce Gestione Cliente . . . . .	37
2.6	Documentazione delle API . . . . .	38
<b>3</b>	<b>Iterazione 2</b>	<b>57</b>
3.1	Algoritmo . . . . .	57
3.1.1	Briefing . . . . .	57
3.1.2	Organizzazione . . . . .	57
3.1.3	Struttura dati . . . . .	58
3.1.4	Funzione di priorità . . . . .	61
3.1.5	Diagramma di flusso . . . . .	64

# Elenco delle figure

1.1	Diagramma dei casi d'uso . . . . .	12
1.2	Topologia del sistema . . . . .	14
2.1	Casi d'uso presi in considerazione nell'iterazione 1 . . . . .	18
2.2	Component diagram - ServeEasy . . . . .	20
2.3	Component diagram - System . . . . .	20
2.4	Architettura esagonale per il microservizio Gestione comanda . . . . .	22
2.5	Component diagram - Gestione Comanda . . . . .	23
2.6	Component diagram - Gestione Comanda - Infrasructure . . . . .	23
2.7	Component diagram - Gestione Comanda - Domain . . . . .	24
2.8	Component diagram - Gestione Comanda - Interface . . . . .	25
2.9	Component diagram - Gestione Cliente . . . . .	26
2.10	Component diagram - Gestione Cliente - Infrastructure . . . . .	27
2.11	Component diagram - Gestione Cliente - Domain . . . . .	28
2.12	Component diagram - Gestione Cliente - Interface . . . . .	28
2.13	Component diagram - Gestione Cucina . . . . .	29
2.14	Component diagram - Gestione Cucina - Infrastructure . . . . .	29
2.15	Component diagram - Gestione Cucina - Domain . . . . .	30
2.16	Component diagram - Gestione Cucina - Interface . . . . .	31
2.17	Modello Entità-relazione . . . . .	32
2.18	Modello Logico . . . . .	33
2.19	Interface class diagram - Gestione Comanda . . . . .	35
2.20	Interface class diagram - Gestione Cucina . . . . .	36
2.21	Interface class diagram - Gestione Cliente . . . . .	37
2.22	Component diagram - Gestione Comanda - Interface con Test . . . . .	38
3.1	Strutture dati dell'algoritmo . . . . .	59
3.2	Diagramma di flusso . . . . .	65



# **Elenco dei codici**

2.1 Query del database in SQL . . . . .	34
---	----



# **Iterazione 0**

## **1.1 Introduzione**

Il sistema che si intende realizzare per il caso di studio è un software gestionale per ottimizzare la gestione delle comande di un ristorante, migliorando l'esperienza dei clienti, la produttività della cucina e l'efficacia della cassa. Il sistema si baserà sull'utilizzo di tablet, che permettono ai commensali di ordinare i piatti desiderati, inserendo eventuali note, visualizzando lo stato degli ordini e richiedere il conto in modo semplice e veloce. La cucina riceve le comande tramite una dashboard dedicata, che le ordina secondo un algoritmo di priorità basato su diversi parametri, come il tempo trascorso dall'ordinazione, la volontà del cliente, la durata di preparazione del piatto e altri fattori. La cucina può anche notificare il completamento di un ordine, che verrà visualizzato sul tablet del tavolo corrispondente. L'operatore di cassa sarà in grado di visualizzare il sommario degli ordini e stampare a schermo una ricevuta al cliente. L'amministratore del ristorante può personalizzare la configurazione delle sale e dei menu, registrare i tavoli e gli account, e visualizzare delle statistiche sulle ordinazioni effettuate. Il sistema offre anche delle funzionalità opzionali, come la possibilità di far arrivare i piatti tutti insieme al tavolo, di allegare note agli ordini in preparazione, chiedere il conto al tavolo. Il sistema si propone quindi di rendere più agile e soddisfacente il servizio di ristorazione, sfruttando le potenzialità della tecnologia e gli alti rendimenti di un algoritmo apposito.

## 1.2 Requisiti funzionali

I requisiti funzionali sono stati esplicitati mediante le *use case stories*, considerando come attori coinvolti nel sistema:

- Amministratore;
- Cliente;
- Cuoco;
- Cassiere.

### 1.2.1 Use case stories: Amministratore

L'amministratore è un responsabile di sala, col compito di configurare il software nelle fasi di setup dell'attività.

#### CONFIGURAZIONE DISPOSITIVI SALA

- Come amministratore, voglio poter registrare i dispositivi destinati ai tavoli dei clienti per consentire ai commensali di accedere al sistema;
- Come amministratore, voglio poter registrare i dispositivi destinati alla cucina per permettere alla cucina di gestire gli ordini;
- Come amministratore, voglio poter registrare un dispositivo destinato al cassiere affinché sia possibile elencare al cliente la comanda che ha ordinato.

#### LOGIN/LOGOUT

- Come amministratore, voglio poter effettuare il log-in/log-out dal sistema.

#### CONFIGURAZIONE MENÙ

- Come amministratore, voglio poter effettuare una gestione del menù per visualizzare/modificare/aggiungere/eliminare portate;
- Come amministratore, voglio poter aggiungere/rimuovere/modificare gli ingredienti assegnati ad una portata per dettagliare la composizione.

### 1.2.2 Use case stories: Cliente

Il cliente può essere di due tipi: il cliente al tavolo, che usufruisce del dispositivo posto a disposizione dal ristorante, e il cliente da asporto, che comunica la sua ordinazione al ristorante tramite un portale sulla rete.

#### AUTENTICAZIONE

- Come cliente, voglio che il sistema riconosca i miei ordini così che possa elaborare le informazioni relative alla mia comanda;

#### VISUALIZZARE MENÙ

- Come cliente, voglio poter visualizzare il menù per decidere quale pietanza ordinare.

#### EFFETTUARE UN'ORDINAZIONE

- Come cliente, voglio effettuare un'ordinazione per ottenere una o più pietanze;
- Come cliente, voglio effettuare un'ordinazione personalizzando la pietanza desiderata per, ad esempio, togliere ingredienti non desiderati.

#### VISUALIZZARE STATO ORDINI

- Come cliente, voglio poter visualizzare lo stato di preparazione dei miei ordini per poter avere un feedback dalla cucina.

#### ANNULLARE UN ORDINE

- Come cliente, voglio annullare l'ordinazione di un piatto.

#### MODIFICARE UN ORDINE

- Come cliente al tavolo, voglio poter modificare un ordine già mandato verso la cucina per, ad esempio, precisare ingredienti da togliere, qualora l'ordine non fosse già in preparazione;
- Come cliente al tavolo, voglio poter modificare un ordine già mandato verso la cucina per, ad esempio, esigere il piatto prima (ad es., se il cliente ritiene di star aspettando troppo) o posticipare la sua preparazione.

### 1.2.3 Use case stories: Cuoco

Il terzo attore coinvolto è il cuoco che prepara le ordinazioni col supporto del sistema.

## GESTIONE PREPARAZIONE ORDINI

- come cuoco, voglio poter gestire gli ordini effettuati dai clienti per poter eventualmente gestire la priorità di essi;
- come cuoco, voglio poter modificare lo stato di un piatto per avvertire il sistema di un'avvenuta preparazione.

## VISUALIZZAZIONE LISTA ORDINI

- Come cuoco, voglio poter verificare lo stato degli ordini richiesti.

### 1.2.4 Use case stories: Cassiere

Il cassiere legge la comanda del cliente al fine di elencare le pietanze da lui ordinate, dettagliare informazioni annesse e calcolarne il conto.

## VISUALIZZARE COMANDA

- Come cassiere, voglio visualizzare la comanda delle ordinazioni relativa a un determinato cliente per generare il conto.

## GENERAZIONE CONTO

- Come cassiere, voglio poter generare il conto per un determinato cliente, per concludere la sua sessione nel sistema.

### 1.2.5 Priorità dei casi d'uso

Per ottimizzare il processo di sviluppo, si è deciso di categorizzare le specifiche funzionali in tabelle con tre livelli di priorità: elevata, media e bassa. Nello specifico il primo livello è assegnato alla Tabella 1.1 a cui sono attribuiti i casi d'uso essenziali per il funzionamento dell'applicazione, i casi d'uso relativi alle funzionalità aggiuntive non critiche sono stati attribuiti alla Tabella 1.2 a priorità media, mentre il livello a bassa priorità che accoglie requisiti funzionali opzionali previsti per versioni successive alla Tabella 1.3 .

## PRIORITÀ ELEVATA

Codice	Titolo
UC1	Gestione comanda
UC2	Effettuare un'ordinazione
UC3	Visualizzare menù
UC4	Autenticazione
UC5	Visualizzare lista ordini
UC6	Gestione preparazione ordini

**Tabella 1.1:** Casi d'uso ad elevata priorità

## PRIORITÀ MEDIA

Codice	Titolo
UC7	Configurazione dispositivi sala
UC8	Gestione dispositivi
UC9	Login amministratore
UC10	Logout amministratore
UC11	Configurazione menù
UC12	Gestione dati menù

**Tabella 1.2:** Casi d'uso a media priorità

## PRIORITÀ BASSA

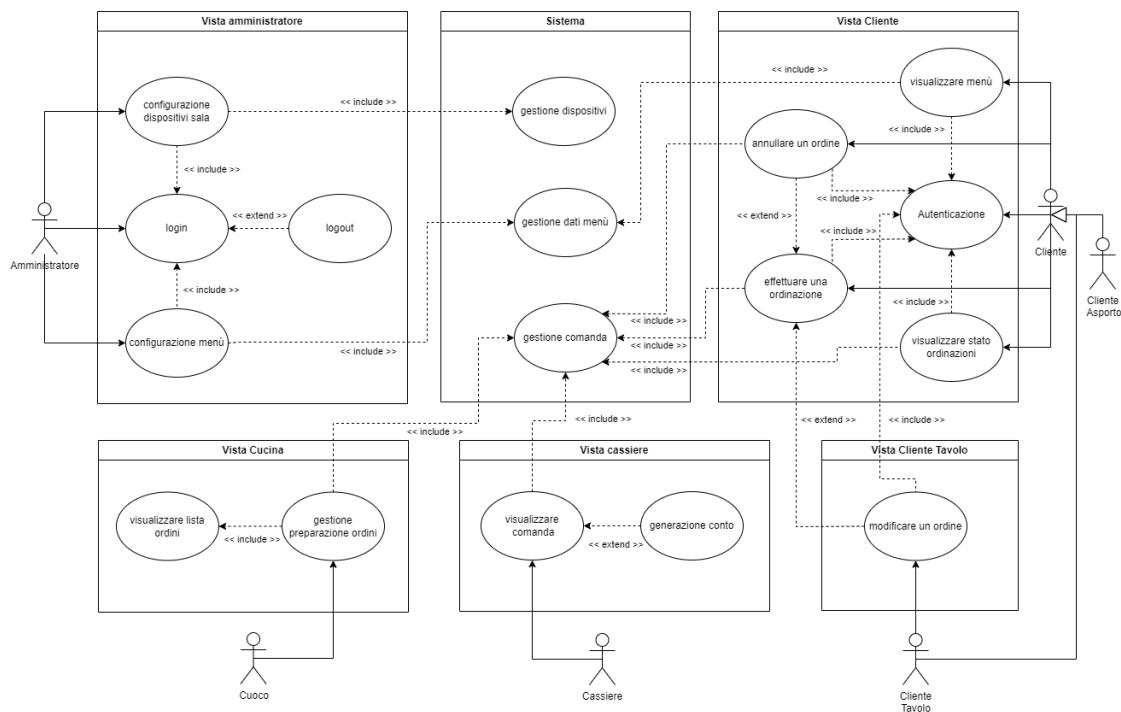
Codice	Titolo
UC13	Modifica un ordine
UC14	Annnullare un ordine
UC15	Visualizza stato delle ordinazioni
UC16	Generazione conto
UC17	Visualizzare comanda

**Tabella 1.3:** Casi d'uso a bassa priorità

### 1.2.6 Use case diagram

Dalla descrizione delle *use case stories*, è stato creato il diagramma UML dei casi d'uso in Figura 1.1, il quale è composto da 4 attori (Amministratore, Cuoco, Cassiere e Cliente) che tramite ereditarietà viene ridefinito in Cliente al tavolo oppure Cliente che effettua

ordinazioni d'asporto) e 6 viste (vista amministratore, vista cucina, vista cassiere, vista cliente, vista cliente al tavolo e sistema).



**Figura 1.1:** Diagramma dei casi d'uso

## 1.3 Requisiti non funzionali

Il progetto verrà sviluppato tenendo considerazione delle performance, integrabilità, modificabilità, testabilità e sicurezza dei componenti.

### 1.3.1 Performance

L'algoritmo di priorità impiegato dalla cucina per la selezione degli ordini deve fornire risultati in un tempo utile. Allo stesso tempo, gli utenti dell'applicativo web devono poter accedere e aggiornare le informazioni in un tempo accettabile.

### 1.3.2 Integrabilità

Ogni componente di sistema deve collaborare con gli altri componenti in modo da garantire le funzionalità previste dal sistema. Questa caratteristica è essenziale per garantire il corretto funzionamento e la coerenza dell'intero sistema.

### 1.3.3 Modificabilità

Il software deve facilitare l'aggiunta di nuovi componenti e funzionalità.

### 1.3.4 Testabilità

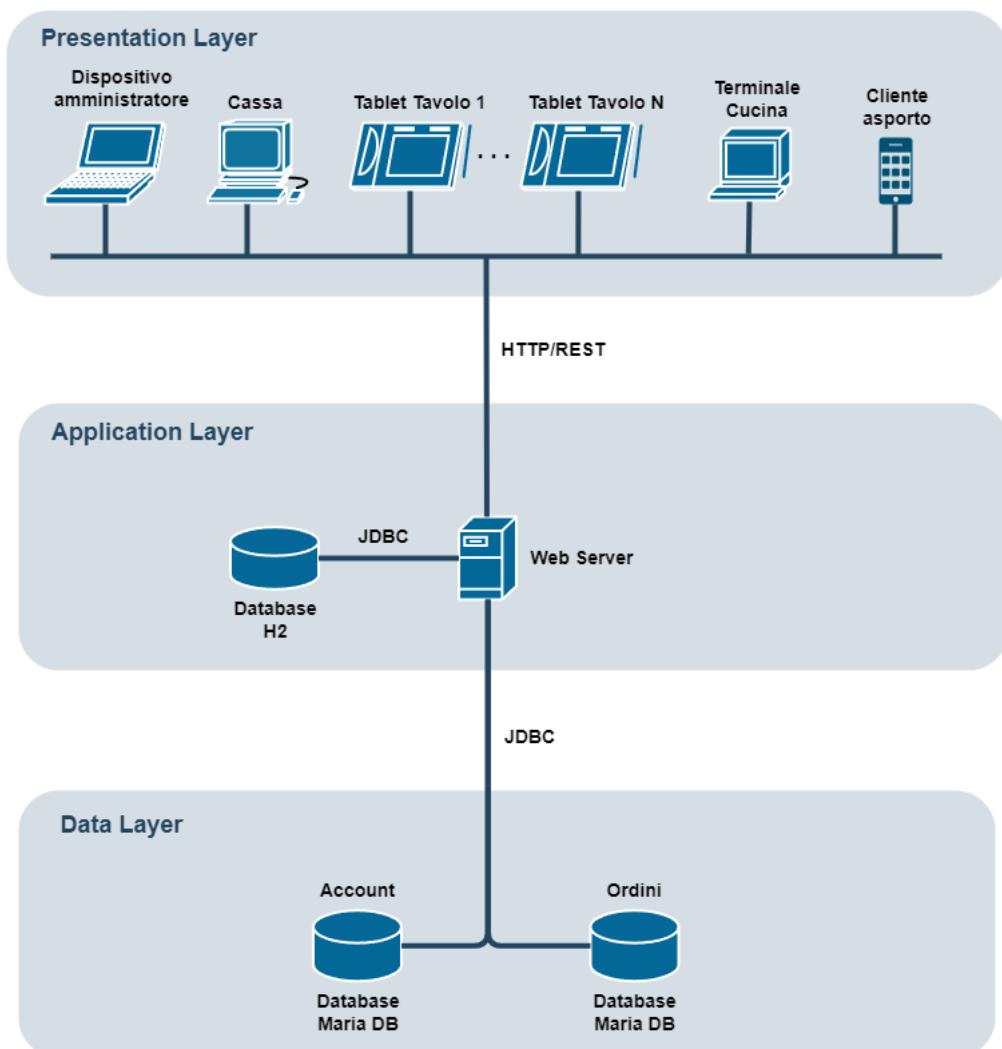
Ogni componente deve poter permettere la progettazione, implementazione ed esecuzione di test efficaci, in modo da garantire una massima copertura di requisiti e funzionalità.

### 1.3.5 Sicurezza

Il sistema deve integrare meccanismi di autenticazione ed autorizzazione degli attori, in modo da garantire la gestione delle identità, oltre alla protezione dei dati e delle API da accessi non autorizzati. Risulta dunque necessaria una distinzione dei ruoli con cui gli attori accedono al sistema.

## 1.4 Topologia

Per il progetto è stata adottata una topologia three-tier al fine di separare in tre livelli distinti la presentazione dei dati, la gestione dell'applicativo e la mappatura dei dati sui dispositivi di archiviazione. Come si può vedere dalla Figura 1.2 il servizio è esposto tramite un web server, al quale i dispositivi clienti accedono, tramite richiesta HTTP/REST, per mezzo di una API unificata, con funzionalità di gateway. Il web-server usufruirà di database relazionali per lo storage (Data Layer), mentre sarà supportato da un database in-memory H2 (Application Layer) per avvantaggiarsi di una ridondanza dati, allo scopo di aumentare le performance lato client.



**Figura 1.2:** Topologia del sistema

## 1.5 Toolchain

Di seguito è presentata la toolchain utilizzata per lo sviluppo del progetto software

### 1.5.1 Modellazione

- draw.io: casi d'uso e topologia;

### 1.5.2 Stack applicativo

- Angular.js: front-end;
- Java Spring Boot 3.x.x: back-end;
- MariaDB: database per l'archiviazione;
- H2: database in-memory per rendere più efficiente l'estrazione dei dati;

### 1.5.3 Deployment

- Docker: piattaforma per container virtuali;
- Docker Compose: gestione app multi-container;

### 1.5.4 Gestore repository

- Git: Controllo versione per codice sorgente;
- GitHub: Piattaforma hosting e collaborativa per progetti Git;

### 1.5.5 Continuous Integration

- Maven: gestore di progetti e dipendenze Java;
- GitHub Action: piattaforma di automazione per repository GitHub;
- Jenkins: strumento di automazione per sviluppatori;

### 1.5.6 Analisi statica

- Checkstyle: visualizzazione di alto livello di metriche qualitative del codice;

### 1.5.7 Analisi dinamica

- Postman: strumento per testare API e servizi;
- Garfana: analisi delle performance della rete di microservizi;
- JUNIT: framework per test unitari Java;

### 1.5.8 Documentazione e organizzazione del team

- Google Drive: servizio cloud per archiviazione;
- Documenti condivisi di Google: per elaborare la documentazione in modo condiviso;
- L<sup>A</sup>T<sub>E</sub>X: generazione documentazione;
- Microsoft Teams: per organizzazione e meeting;

### 1.5.9 Modello di sviluppo

Il modello adottato segue la filosofia AGILE, con enfasi sui seguenti aspetti-chiave:

- pair programming, per favorire creatività e controllo del lavoro prodotto;
- orientamento al risultato, con enfasi maggiore sulla generazione di codice funzionante e componenti completi prima della relativa documentazione;
- rapidità di risposta ai cambiamenti;
- collaborazione attiva col cliente, al fine di incontrare le sue necessità, garantire trasparenza e fornire feedback tempestivo sul lavoro di progetto;
- Proattività nell'identificazione e mitigazione dei rischi.

# **Iterazione 1**

## **2.1 Introduzione**

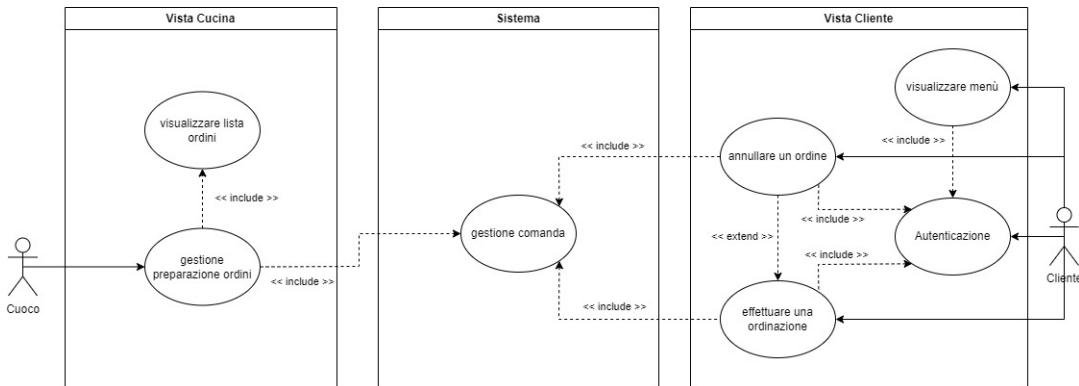
Nella Iterazione 1 si sono presi i casi d'uso a più alta priorità e ci si è focalizzati allo sviluppo della architettura software, del database e dell'algoritmo. Si è adottato un approccio di good design, puntando a un sistema software di alta qualità, mantenibile e scalabile, con componenti modulari e codice chiaro. Parallelamente, si è perseguito il principio di coesione funzionale, assicurando che funzioni correlate fossero raggruppate per formare moduli coesi, migliorando così manutenibilità e testabilità del sistema. E' stato quindi eseguito un lavoro di analisi e decomposizione del problema seguendo delle euristiche di early design, riunendo gli use cases in gruppi a cui potessero essere associati dei subsystem ben delineati. L'applicazione delle euristiche assume che la progettazione della soluzione si baserà su un'architettura a microservizi.

## 2.2 Use Cases

Si sono presi in considerazione i seguenti casi d'uso, ossia quelli a priorità più elevata della Tabella 1.1

Codice	Titolo
UC1	Gestione comanda
UC2	Effettuare un'ordinazione
UC3	Visualizzare menù
UC4	Autenticazione
UC5	Visualizzare lista ordini
UC6	Gestione preparazione ordini

**Tabella 2.1:** Casi d'uso presi in considerazione nell'iterazione 1



**Figura 2.1:** Casi d'uso presi in considerazione nell'iterazione 1

Per facilitare una migliore organizzazione e comprensione del sistema, i casi d'uso vengono raggruppati nel seguente modo:

### 2.2.1 Gruppo Sistema

#### UC-1 “Gestione Comanda”:

- UC-1.1 : gestione priorità ordine  
ogni ordine è caratterizzato da una priorità
- UC-1.2 : gestione coda ordini  
ogni ordine è inserito in una coda ordini
- UC-1.3 : assegnazione ordini comanda  
ogni ordine deve essere associato ad una comanda

- UC-1.4 : assegnazione comanda cliente  
ogni comanda deve essere associata ad un cliente

## 2.2.2 Gruppo cliente

### UC-2 “effettuare un’ordinazione”:

- UC-2.1 : effettuare un ordine personalizzato  
il cliente può effettuare un ordine escludendo un ingrediente o descrivendo una variazione del piatto

### UC-3 “Visualizzare menu”:

- UC-3.1 : visualizzare piatto
- UC-3.2 : visualizzare informazioni piatto  
il cliente deve poter leggere breve descrizione, ingredienti, prezzo

### UC-4 “Autenticazione”:

- UC-4.1 : identificazione sessione cliente  
al momento del pasto e solo per il pasto, il cliente deve poter distinguere la propria comanda

## 2.2.3 Gruppo cuoco

### UC-5 “visualizzare lista ordini”:

- UC-5.1 : visualizzazione ordini per postazione  
il cuoco deve visualizzare gli ordini destinati alla sua postazione

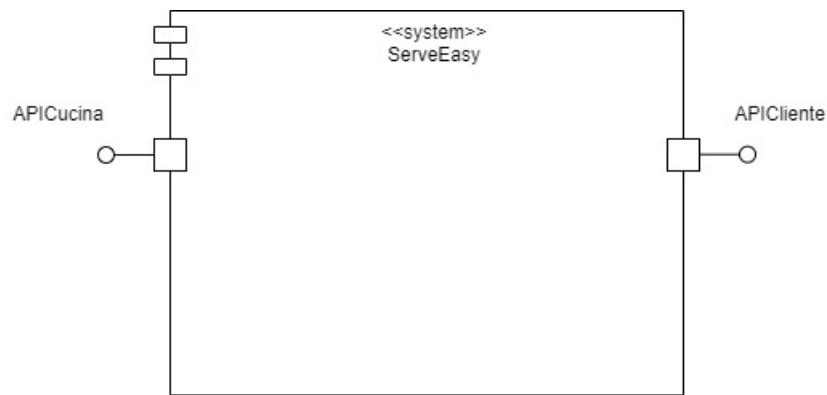
### UC-6 “gestione preparazione ordini”:

- UC-6.1 : notifica preparazione ordine  
il cuoco deve segnalare la presa in carico dell’ordine
- UC-6.2 : notifica completamento ordine  
il cuoco deve segnalare il completamento dell’ordine così da passare al successivo
- UC-6.3 : gestione priorità postazione  
il cuoco può modificare la priorità di un certo ingrediente così da ridurre la pressione su una certa postazione o, viceversa, per aumentarne il traffico. In tal modo può manualmente agire sulla gestione del traffico verso la cucina.

## 2.3 Component Diagram

### 2.3.1 Sistema ServeEasy

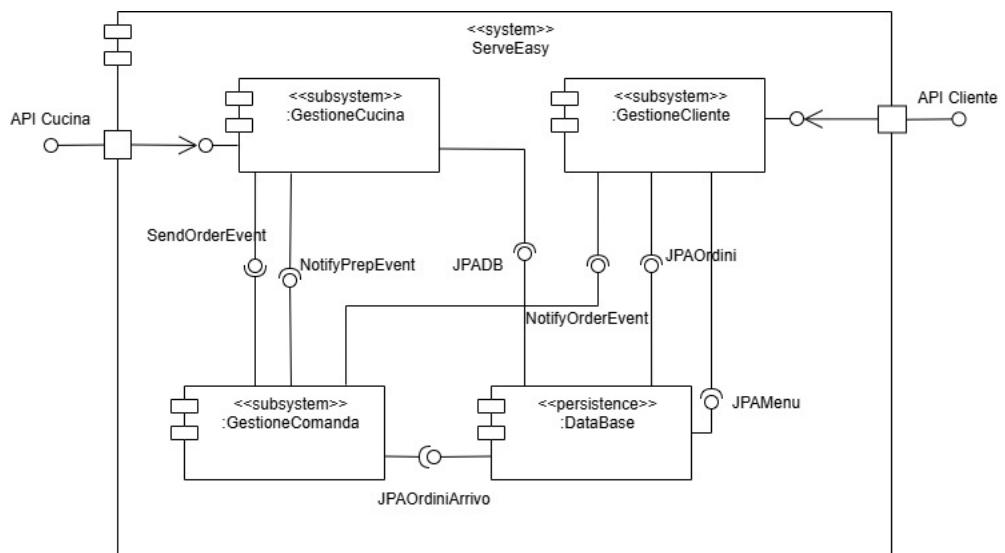
single component del sistema



**Figura 2.2:** Component diagram - ServeEasy

Visualizzazione iniziale della soluzione come un componente unico che espone due API, dedicate rispettivamente alla cucina ed ai clienti. Si procede con uno sviluppo top-down.

primo zoom-in sul sistema



**Figura 2.3:** Component diagram - System

Al primo zoom-in si identificano i servizi che andranno a comporre l'architettura della soluzione:

- **GestioneComanda:** risolve gli use case del gruppo “sistema”, rappresenta il cuore del sistema ed incorpora la logica di backend fondamentale per la gestione regolarizzata degli ordini da cliente a cucina, attraverso politiche di schedulazione a priorità progettate ed implementate con un algoritmo ad-hoc.
- **GestioneCliente:** risolve gli use case del gruppo “cliente”, espone le funzionalità destinate ai dispositivi di tavolo ed al portale web per clienti d’asporto. Ha dunque il compito di gestire gli aspetti del servizio legati alle interazioni del cliente col sistema, come la visualizzazione del menu, la creazione degli ordini ed il raggruppamento degli ordini in una comanda relativa.
- **GestioneCucina:** risolve gli use case del gruppo “cucina”, espone le chiamate destinate ai dispositivi di cucina. Questo servizio conterrà un sistema a code, dove l’ordine in arrivo verrà classificato ed inserito in base al suo ingrediente principale. Gli ordini verranno gestiti dalle postazioni della cucina seguendo una politica FIFO.

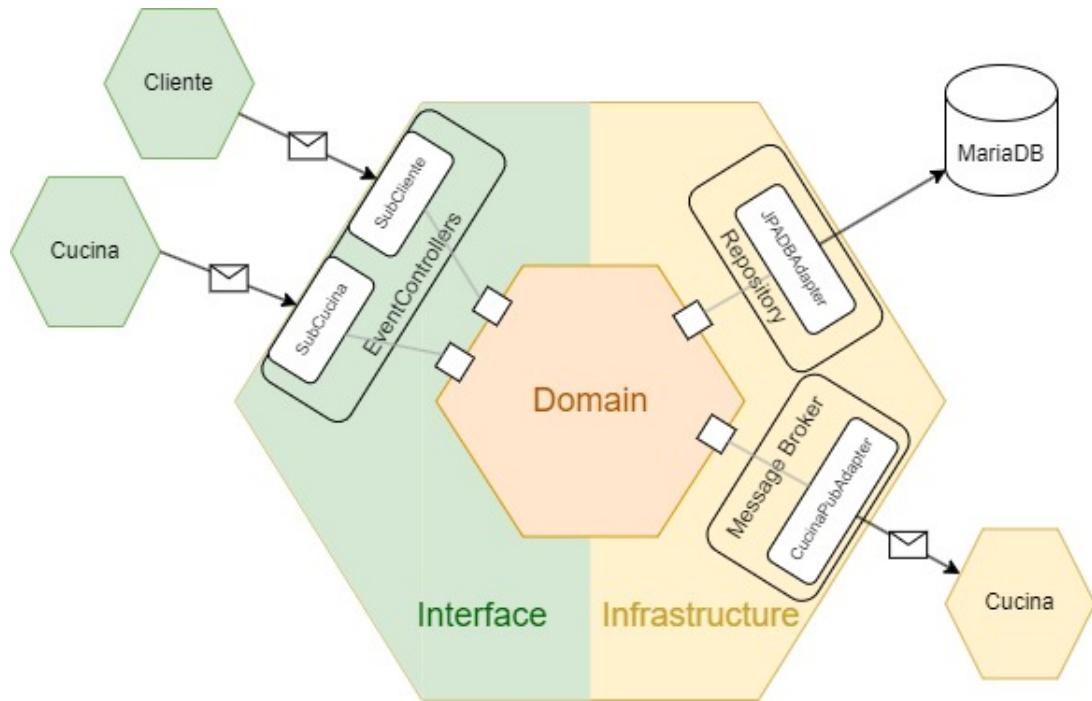
Per la memorizzazione persistente dei dati cruciali per l’attività come piatti, ordini e comande, è stato inserito un componente database. All’interno del sistema ServeEasy, i componenti comunicano tra loro attraverso una comunicazione ad eventi, asincrona. Si è deciso di attuare una politica pub-sub per la gestione delle comunicazioni interne, costituite da scambi di notifiche e DTO tra i microservizi designati.

### 2.3.2 Gestione Comanda

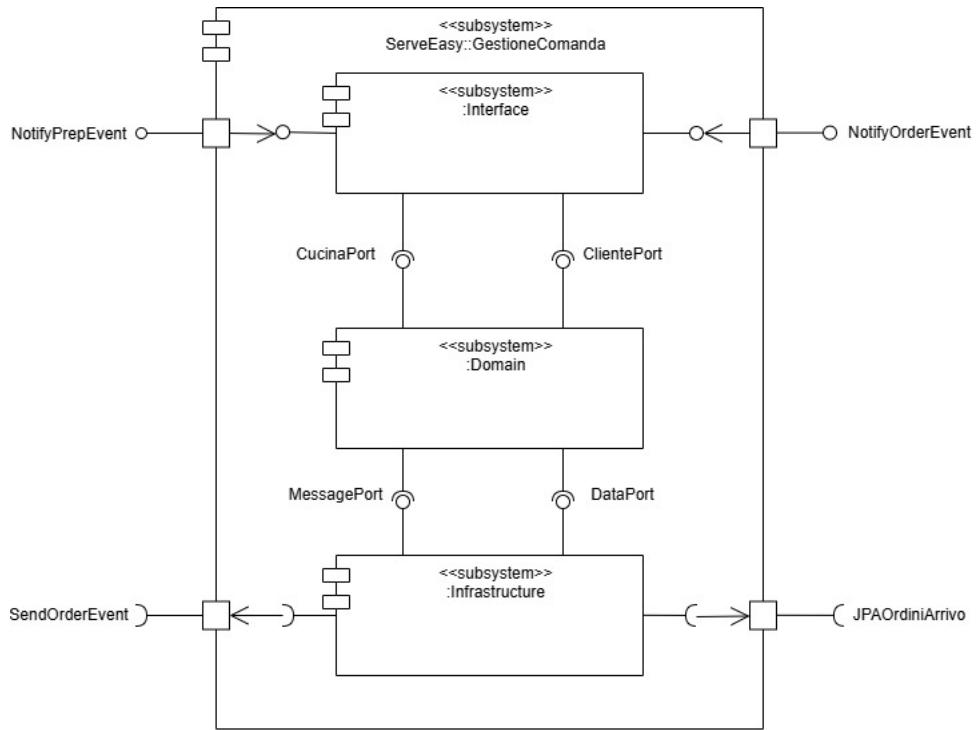
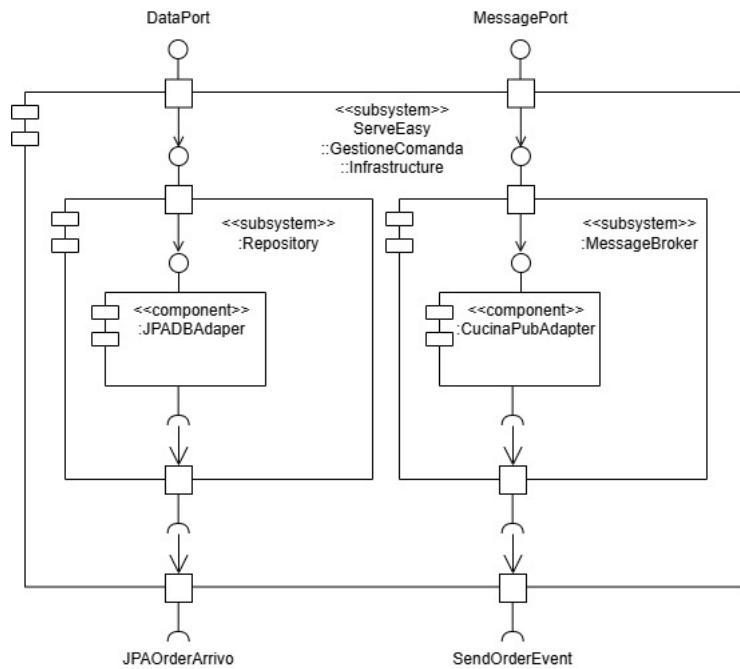
#### Componenti esagonali

Il design dei microservizi seguirà l’architettura esagonale: un dominio, denominato “Domain”, nucleo della logica di servizio, sarà racchiuso tra due gusci denominati “Interface” e “Infrastructure”, i quali avranno il compito di astrarre la gestione dati, rendendola opaca al dominio. La logica di base del microservizio seguirà lo schema port-adapter, dove il dominio comunica con i gusci attraverso delle interfacce dette porte (il cui nome nel progetto è caratterizzato dal suffisso “Port”), mentre i gusci hanno il compito di implementare l’effettivo componente di trasmissione (guscio Infrastructure) e/o ricezione (guscio Interface), detto adattatore (sarà identificabile da suffisso “Adapter”). Nello specifico il **Domain** definisce gli oggetti, le entità e le operazioni che sono pertinenti al problema che il microservizio gestisce. Gli **Interface adapters** fungono da ponte tra il mondo esterno e il core del sistema, consentendo al microservizio di comunicare con altre applicazioni, servizi o dispositivi esterni in modo indipendente dall’implementazione interna del sistema stesso, mentre gli **Infrastructure adapters** fungono da ponte tra il core del sistema e l’infrastruttura esterna, gestendo le chiamate e le operazioni necessarie per accedere e

utilizzare le risorse infrastrutturali. Tali caratteristiche avvantaggiano l'intercambiabilità dei singoli componenti di sistema a costo di un aumento della complessità.

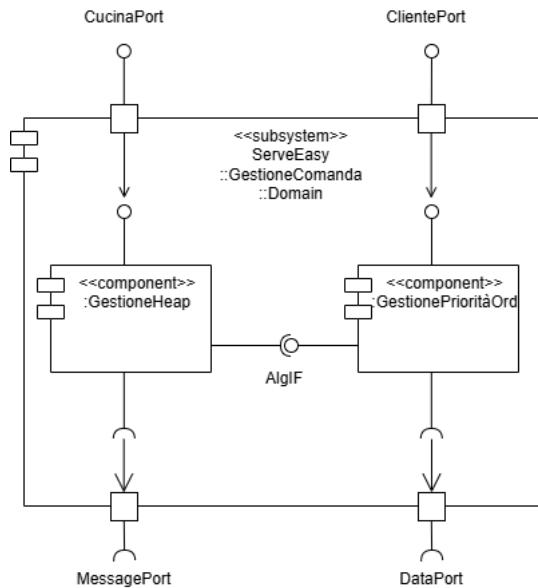


**Figura 2.4:** Architettura esagonale per il microservizio Gestione comanda

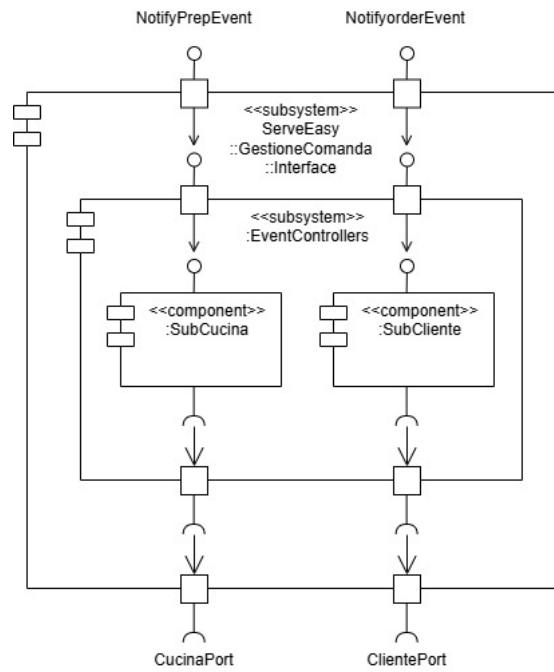
**zoom-in gestione comanda****Figura 2.5:** Component diagram - Gestione Comanda**zoom-in infrastruttura di gestione comanda****Figura 2.6:** Component diagram - Gestione Comanda - Infrastruttura

- Repository: JPADBAdapter per la comunicazione con il database;
- MessageBroker: CucinaPubAdapter per l'invio di messaggi sul topic verso il microservizio della cucina.

### zoom-in domain di gestione comanda



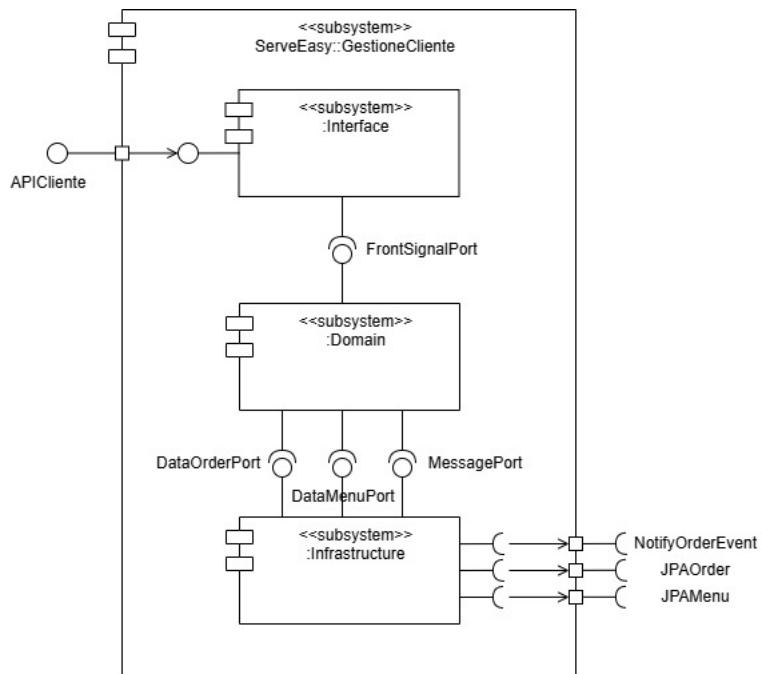
**Figura 2.7:** Component diagram - Gestione Comanda - Domain

**zoom-in interface di gestione comanda****Figura 2.8:** Component diagram - Gestione Comanda - Interface

- EventControllers: `SubCucina` e `SubCliente`, permettono la ricezione di messaggi tramite message broker dagli altri microservizi.

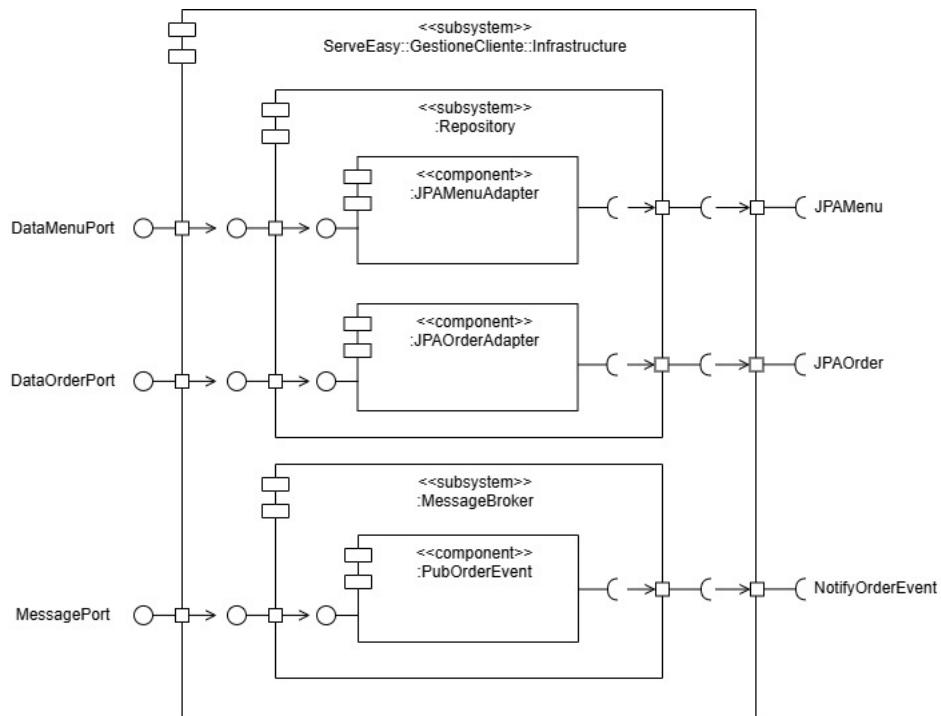
### 2.3.3 Gestione Cliente

zoom-in gestione cliente



**Figura 2.9:** Component diagram - Gestione Cliente

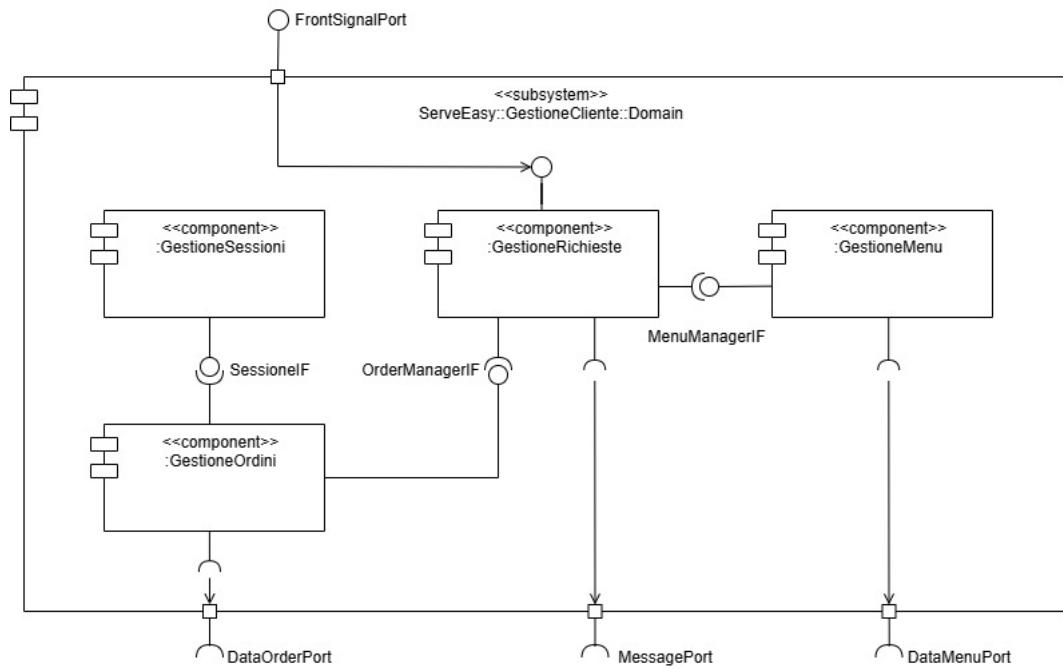
### zoom-in infrastructure di gestione cliente



**Figura 2.10:** Component diagram - Gestione Cliente - Infrastructure

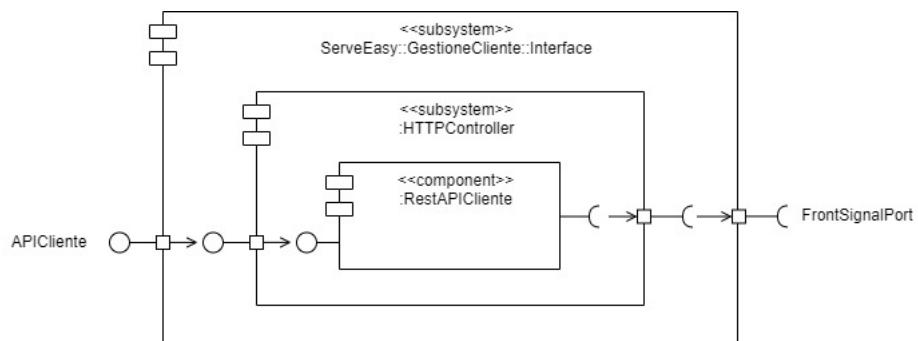
- Repository: **JPAOrderAdapter** e **JPAMenuAdapter** per la comunicazione con il database;
- MessageBroker: **PubOrderEvent** per l'invio di messaggi sul topic verso il microservizio della cucina.

### zoom-in domain di gestione cliente



**Figura 2.11:** Component diagram - Gestione Cliente - Domain

### zoom-in interface di gestione cliente

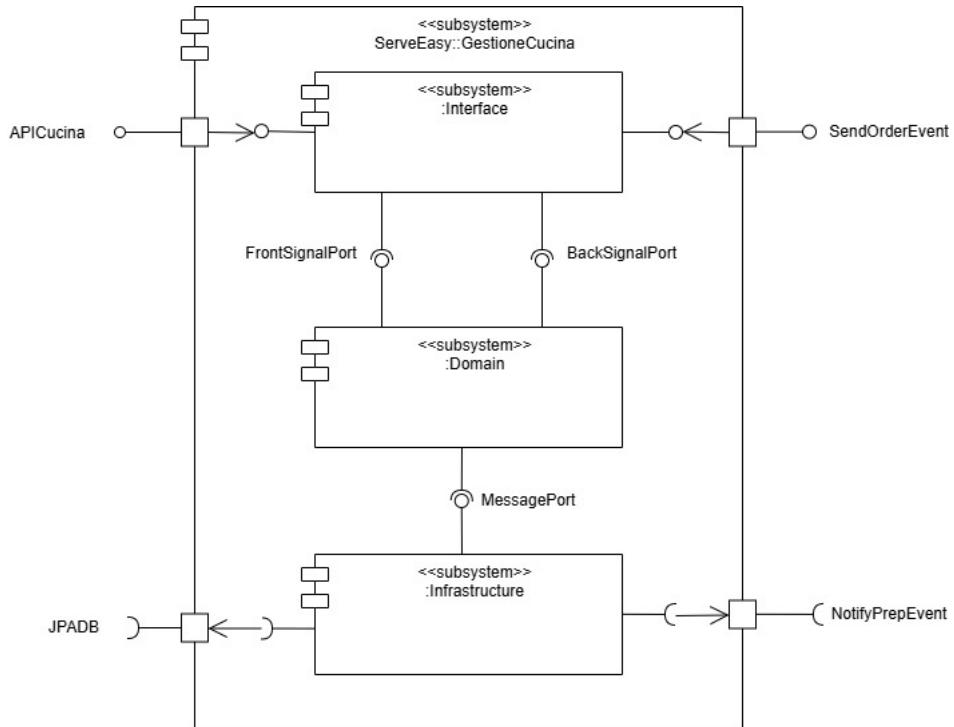


**Figura 2.12:** Component diagram - Gestione Cliente - Interface

- `HTTPControllers: RestApiClient`, permette di esporre API verso l'esterno.

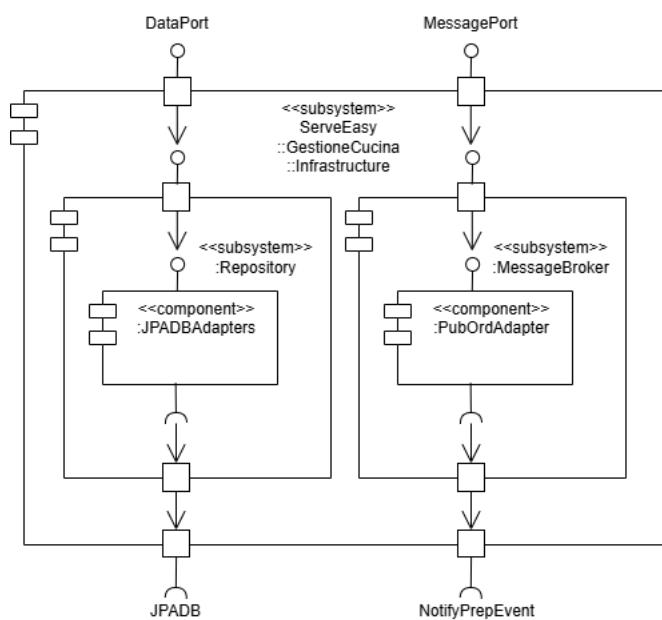
### 2.3.4 Gestione Cucina

#### zoom-in gestione cucina



**Figura 2.13:** Component diagram - Gestione Cucina

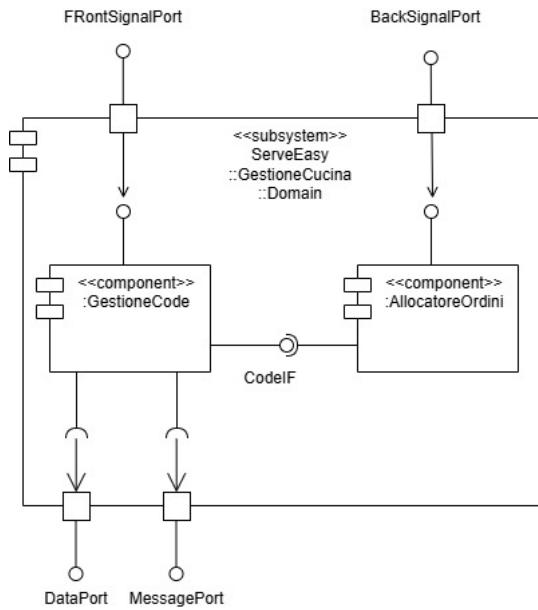
#### zoom-in infrastruttura di gestione cucina



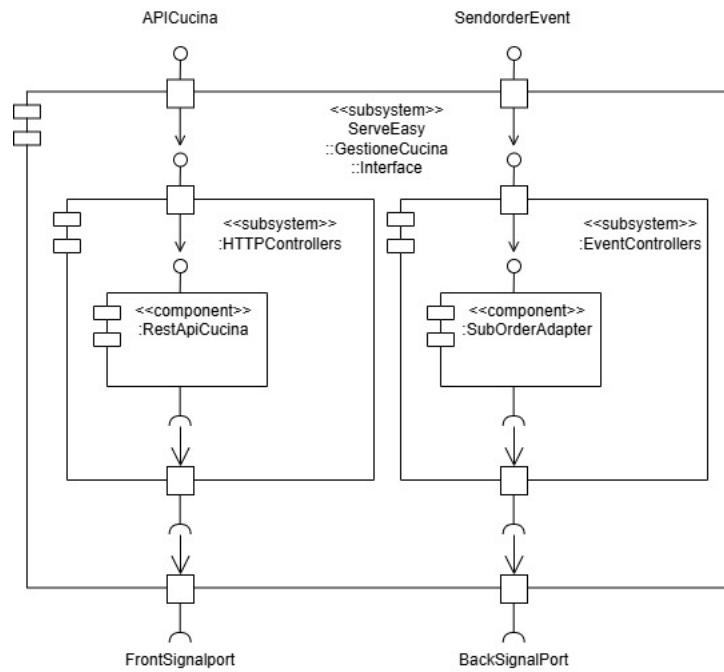
**Figura 2.14:** Component diagram - Gestione Cucina - Infrastructure

- Repository: JPADBAdapter per la comunicazione con il database;
- MessageBroker: PubOrderAdapter per l'invio di messaggi sul topic verso il micro-servizio di GestioneComanda.

### zoom-in domain di gestione cucina



**Figura 2.15:** Component diagram - Gestion Cucina - Domain

**zoom-in interface di gestione cucina****Figura 2.16:** Component diagram - Gestione Cucina - Interface

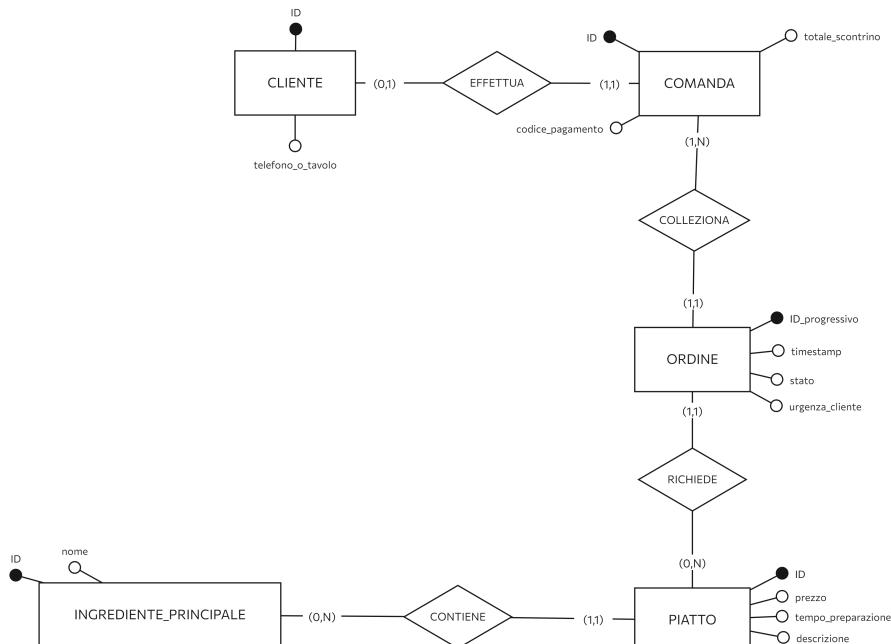
- EventControllers: SubOrderAdapter, permette la ricezione di messaggi tramite message broker dal microservizio GestioneComanda;
- HTTPControllers: RestApiCucina, permette di esporre API verso l'esterno.

## 2.4 Database

Per facilitare l'identificazione delle entità coinvolte nel database si è utilizzato un modello entità-relazione che fornisce una rappresentazione grafica chiara e intuitiva della struttura dei dati. Questo modello aiuta a visualizzare le entità (oggetti o concetti del mondo reale), le relazioni (le associazioni tra le entità) e gli attributi (le proprietà o le caratteristiche delle entità e delle relazioni).

### 2.4.1 Modello Entità-Relazione

Nel seguente diagramma entità-relazione in Figura 2.17, osserviamo che le comande possono essere costituite da più ordini effettuati dai clienti. Tali clienti sono suddivisi in due categorie: clienti d'asporto identificati tramite numero di telefono e clienti al tavolo identificati tramite numero del tavolo. I piatti, consultabili tramite un menù, sono caratterizzati da un ingrediente principale. Una volta ordinato un piatto dal menù, questo viene inserito al'interno di un ordine identificato da un codice progressivo per cliente, e viene successivamente inserito nella comanda del rispettivo cliente. La comanda sarà quindi utilizzata per identificare il cliente e contiene i piatti ordinati oltre che il totale dello scontrino con il corrispettivo codice di pagamento.



**Figura 2.17:** Modello Entità-relazione

## 2.4.2 Modello logico

Tramite il modello logico viene rappresentata in modo astratto la struttura dei dati così da facilitare la progettazione del database, definendo come i dati sono organizzati e come le entità interagiscono tra loro. Rappresentazione della struttura dei dati all'interno del database. L'attributo di cliente::asporto\_o\_tavolo è stato pensato come un boolean in quanto il cliente può essere di due tipi:

- se asporto\_o\_tavolo = 0, allora l'ID sarà il codice identificativo di un tavolo;
- se asporto\_o\_tavolo = 1, allora l'ID sarà un numero di telefono;

Cliente	
PK	ID : varchar
asporto_o_tavolo : bool	

Comanda	
PK	ID_incr : int
FK	ID_cliente
	codice_pagamento : varchar
	totale_scontrino : float

Ordine	
PK	ID_incr : int
FK	ID_piatto
	stato : string
	urgenza_cliente : int
	t_ordinazione : time

IngredientePrincipale	
PK	ID : varchar
nome : string	

Piatto	
PK	ID : varchar
FK	ID_ingr_princ
	descrizione : varchar
	prezzo : float
	t_preparazione : time

**Figura 2.18:** Modello Logico

---

Il modello logico è implementato con le seguenti query al database:

```

1  CREATE TABLE IF NOT EXISTS Cliente(
2    ID varchar(10) PRIMARY KEY,
3    t_o_a boolean NOT NULL
4  );
5
6  CREATE TABLE IF NOT EXISTS Comanda (
7    ID int(10) AUTO_INCREMENT,
8    ID_cliente varchar(10) NOT NULL,
9    codice_pagamento varchar(255) DEFAULT NULL,
10   totale_scontrino float DEFAULT 0.0,
11   PRIMARY KEY (ID),
12   FOREIGN KEY (ID_cliente) REFERENCES Cliente(ID)
13 );
14
15  CREATE TABLE IF NOT EXISTS IngredientePrincipale(
16    ID varchar(20) PRIMARY KEY,
17    nome varchar(20) NOT NULL
18 );
19
20  CREATE TABLE IF NOT EXISTS Piatto(
21    ID varchar(20) NOT NULL PRIMARY KEY,
22    ID_ingr_princ varchar(20) NOT NULL,
23    descrizione varchar(50),
24    prezzo float(6) NOT NULL,
25    t_preparazione TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
26    FOREIGN KEY (ID_ingr_princ) REFERENCES IngredientePrincipale(ID)
27 );
28
29  CREATE TABLE IF NOT EXISTS Ordine(
30    ID int(10) NOT NULL AUTO_INCREMENT,
31    ID_comanda int(10) NOT NULL,
32    ID_piatto varchar(20) NOT NULL,
33    stato int(1) DEFAULT 0, -- 0=in preparazione, 1=completato
34    t_ordinazione TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
35    urgenza_cliente int(2) DEFAULT 0, -- priorita' del cliente: 1=massima
36      , -1=minima
37    PRIMARY KEY (ID, ID_comanda),
38    FOREIGN KEY (ID_comanda) REFERENCES Comanda(ID),
39    FOREIGN KEY (ID_piatto) REFERENCES Piatto(ID),
40    CHECK (stato >= 0 AND stato <=1 )
41 );

```

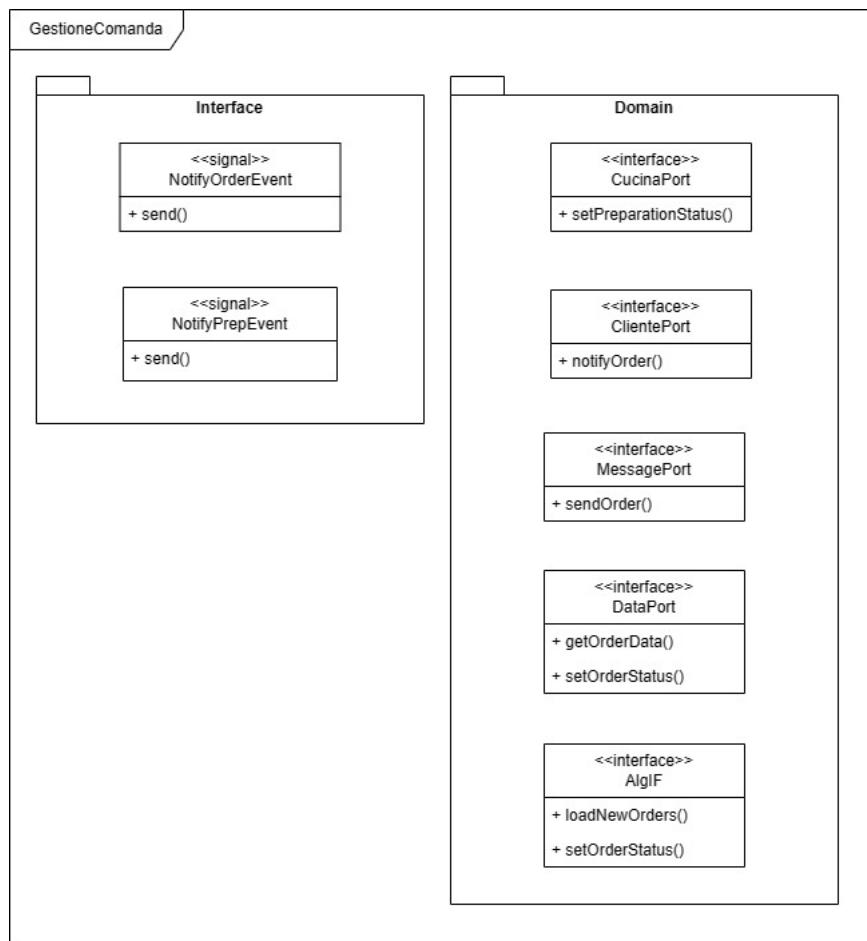
---

**Codice 2.1:** Query del database in SQL

## 2.5 Interface Class Diagram

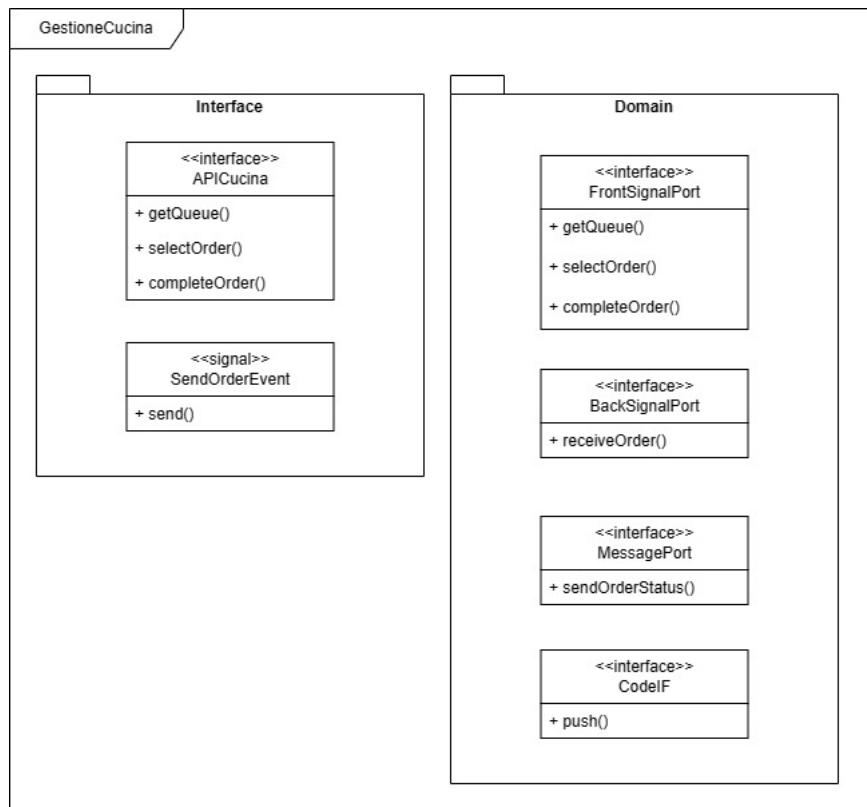
In questa sezione si definiscono in alto livello le interfacce, con relativi metodi, per la comunicazione tra i componenti e sottosistemi ottenuti. In particolare, vengono definiti con lo stereotipo **signal** i canali di comunicazione ad eventi, designati nel progetto per seguire un pattern publisher-subscriber.

### 2.5.1 Interfacce Gestione Comanda



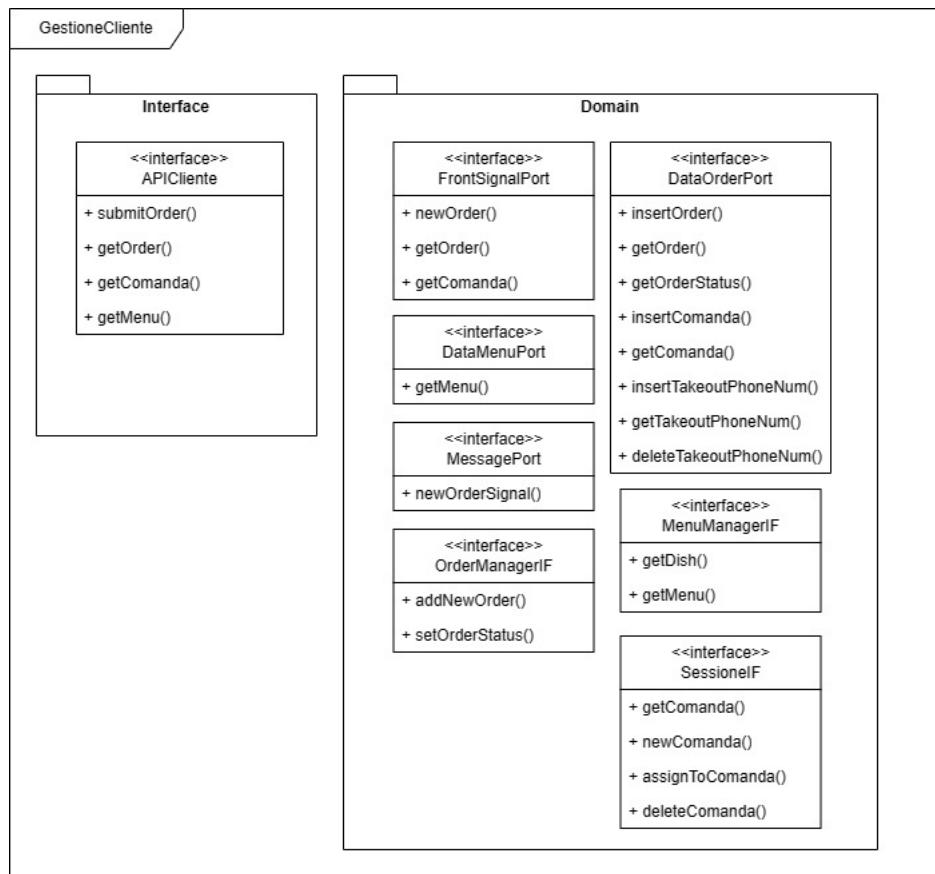
**Figura 2.19:** Interface class diagram - Gestione Comanda

## 2.5.2 Interfacce Gestione Cucina



**Figura 2.20:** Interface class diagram - Gestione Cucina

### 2.5.3 Interfacce Gestione Cliente



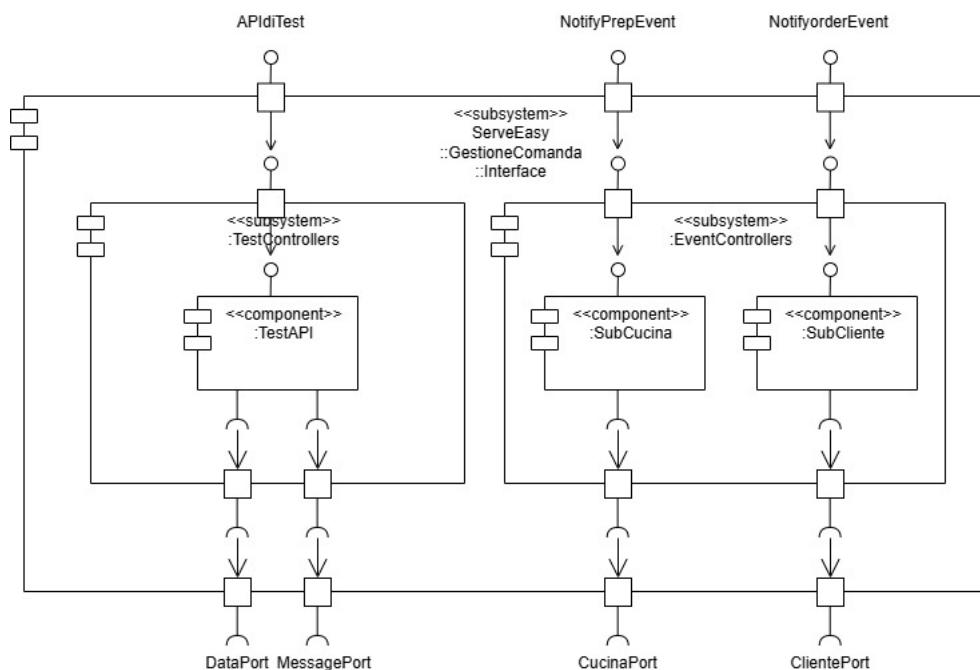
**Figura 2.21:** Interface class diagram - Gestione Cliente

## 2.6 Documentazione delle API

Le API esposte dai microservizi sono state testate tramite [postman](#) utilizzato in locale tramite [postman agent](#) creando un workspace condiviso tra il team.

Il microservizio GestioneComanda è sprovvisto di un componente HTTP Controller nella sua Interfaccia (che contiene solo EventController), viene quindi creato un controller di TEST per interagire direttamente con i componenti del servizio ai soli fini di test.

### zoom-in interface di gestione comanda



**Figura 2.22:** Component diagram - Gestione Comanda - Interface con Test

- EventControllers: SubCucina e SubCliente, permettono la ricezione di messaggi tramite message broker dagli altri microservizi.
- TestControllers: TestAPI per poter testare le API di Test utilizzando direttamente la DataPort e la MessagePort

Viene di seguito allegata la documentazione delle API redatta utilizzando lo strumento [documenter.getpostman](#), link ai documenti ufficiali con anche esempi:

- Gestione Comanda: <https://documenter.getpostman.com/view/32004409/2sA3JDhkaG>
- Gestione Cliente: <https://documenter.getpostman.com/view/32004409/2sA3JFBQDv>
- Gestione Cucina: <https://documenter.getpostman.com/view/32004409/2sA3JF9iav>

## Gestione Comanda

Il microservizio Gestione Comanda si occupa principalmente di gestire gli ordini dei clienti (microservizio GestioneCliente) e fornire alla cucina (microservizio GestioneCucina) gli ordini da preparare.

La comunicazione con gli altri microservizi avviene tramite Message Broker come segue:

- Il microservizio GestioneCliente comunica verso GestioneComanda tramite il topic Kafka NotifyOrderEvent.
- Il microservizio GestioneComanda comunica verso GestioneCucina tramite il topic Kafka SendOrderEvent.
- Il microservizio GestioneCucina comunica verso GestioneComanda tramite il topic Kafka NotifyPrepEvent.

Il microservizio GestioneComanda è sprovvisto di un componente HTTP Controller nella sua Interfaccia (che contiene solo EventController), viene quindi creato un controller di TEST per interagire direttamente con i componenti del servizio ai soli fini di test.

Le API di test riguardano principalmente il Message Broker e il DataBase.

---

### Message Broker

E' possibile interagire direttamente con i tre topic (NotifyOrderEvent, SendOrderEvent, NotifyPrepEvent) tramite operazioni di GET e di POST:

- GET: le chiamate GET all'indirizzo `.../test/{topic}` restituiscono l'ultimo messaggio passato sul topic specificato;
  - POST: le chiamate POST all'indirizzo `.../test/{topic}` permettono di iniettare dall'esterno dei messaggi sul topic specificato, necessitano di un corpo in formato JSON che equivale alla serializzazione dell'oggetto che si aspetta quel topic.
- 

#### POST Post to topic sendOrderEvent

localhost:8080/test/sendorderevent

API di POST con la quale è possibile iniettare all'interno del broker oggetti al fine di test.

Si testa il topic sendOrderEvent da gestione comanda verso gestione cucina.

**Parametri della richiesta (body JSON):**

- `id` (numero intero, opzionale) : Identificatore dell'ordine (autogenerato in ogni caso dal sistema)
- `idComanda` (numero intero, obbligatorio) : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` (stringa, obbligatorio) : Identificatore del piatto ordinato dal cliente
- `stato` (numero intero tra 0 e 3, obbligatorio) : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` (stringa timestamp di pattern "yyyy-MM-dd HH:mm:ss.SSS", opzionale): Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` (numero intero tra 0 e 2, obbligatorio) : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

**Body raw (json)**

json

```
{
  "idComanda":7,
  "idPiatto":"SUH724",
  "stato":1,
  "urgenzaCliente":0
}
```

**GET Get from topic sendOrderEvent**

localhost:8080/test/sendorderevent

API di GET con la quale è possibile ottenere l'ultimo messaggio letto sul topic SendOrderEvent.

Si testa il topic notifyPrepEvent da gestione cucina verso gestione comanda

**Risposta:**

json

```
{  
    "id": 7,  
    "idComanda": 7,  
    "idPiatto": "SUH724",  
    "stato": 1,  
    "urgenzaCliente": 0,  
    "tordinazione": "2024-04-30 22:03:17.199"  
}
```

## POST Post to topic notifyOrderEvent

localhost:8080/test/notifyorderevent

API di POST con la quale è possibile iniettare all'interno del broker oggetti al fine di test.

Si testa il topic notifyOrderEvent da gestione cliente verso gestione comanda.

**Parametri della richiesta (body JSON):**

- `id` (numero intero, obbligatorio) : Identificatore dell'ordine
- `idComanda` (numero intero, obbligatorio) : Identificatore della comanda di cui l'ordine fa parte

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte

**Body** raw (json)

json

```
{  
    "id": 1,  
  
    "idComanda": 4  
}
```

## GET Get from topic notifyOrderEvent

localhost:8080/test/notifyorderevent

Espone una API di GET con la quale è possibile ottenere l'ultimo messaggio letto sul topic NotifyOrderEvent.

Si testa il topic notifyPrepEvent da gestione cucina verso gestione comanda.

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte

## POST Post to topic notifyPrepEvent

localhost:8080/test/notifyprepevent

API di POST con la quale è possibile iniettare all'interno del broker oggetti al fine di test.

Si testa il topic notifyPrepEvent da gestione cucina verso gestione comanda.

**Parametri della richiesta (body JSON):**

- `id` (numero intero, obbligatorio) : Identificatore dell'ordine
- `idComanda` (numero intero, obbligatorio) : Identificatore della comanda di cui l'ordine fa parte
- `stato` (numero intero tra 0 e 3, obbligatorio) : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in

preparazione, 3: Ordine preparato)

### Body raw (json)

json

```
{  
    "id": 1,  
    "idComanda": 4,  
    "stato": 2  
}
```

## GET Get from topic notifyPepEvent

localhost:8080/test/notifyprepevent

API di GET con la quale è possibile ottenere l'ultimo messaggio letto sul topic NotifyPreEvent.

Si testa il topic notifyPreEvent da gestione cucina verso gestione comanda.

### Risposta:

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)

## Database

Le API di test verso il database riguardano unicamente l'entità Ordine, vengono testati:

- l'inserimento nel database;
- la ricerca di un elemento;
- la modifica parziale di un elemento;
- la ricerca di tutti gli ordini per una data comanda;
- la rimozione di un ordine

## POST Post order

localhost:8080/test/order

Salva nel database l'oggetto ordine dato un ordineDTO

Parametri della richiesta (body JSON):

- `id` (numero intero, opzionale) : Identificatore dell'ordine (autogenerato in ogni caso dal sistema)
- `idComanda` (numero intero, obbligatorio) : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` (stringa, obbligatorio) : Identificatore del piatto ordinato dal cliente
- `stato` (numero intero tra 0 e 3, obbligatorio) : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` (stringa timestamp di pattern "yyyy-MM-dd HH:mm:ss.SSS", opzionale): Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` (numero intero tra 0 e 2, obbligatorio) : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

Risposta:

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

Body raw (json)

json

```
{  
    "idComanda":4,  
    "idPiatto":"PIA770",  
    "stato":0,  
    "urgenzaCliente":1
```

3

## GET Get order by id

```
localhost:8080/test/order/1
```

Restituisci l'ordine corrispondente all'id dato in input

**Parametri della query string:**

- `id` (obbligatorio): Identificatore dell'ordine da recuperare

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

## PATCH Patch order

```
localhost:8080/test/order/1
```

Aggiornamento parziale dell'entità ordine, è possibile fornire solamente gli oggetti da aggiornare.

**Parametri della query string:**

- `id` (obbligatorio): Identificatore dell'ordine da recuperare

**Parametri della richiesta (body JSON):**

- `stato` (numero intero tra 0 e 3, opzionale) : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `urgenzaCliente` (numero intero tra 0 e 2, opzionale) : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

**Body** raw (json)

json

```
{  
  "stato": 1,  
  "urgenzaCliente": 1  
}
```

**GET Get orders by IdComanda**

localhost:8080/test/orders/4

Restituisce una lista con tutti gli ordini relativi a una data comanda

**Parametri della query string:**

- `idComanda` (obbligatorio): Identificatore della comanda di cui gli ordini fanno parte

**Risposta:**

Lista di oggetti con i seguenti parametri:

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

---

## DELETE Delete order

localhost:8080/test/order/8

Cancella l'ordine con il dato ID dal database

**Parametri della query string:**

- `id` (obbligatorio): Identificatore dell'ordine da recuperare

**Risposta:**

Plain Text

204 No Content

## Gestione Cliente

Il microservizio Gestione Cliente si occupa principalmente di gestire l'interazione di un cliente col sistema e di gestire la sua relativa comanda di ordini.

La comunicazione con gli altri microservizi avviene tramite Message Broker come segue:

- Il microservizio GestioneCliente comunica verso GestioneComanda tramite il topic Kafka NotifyOrderEvent.

Le API riguardano l'interazione del cliente con il servizio

---

### GET Get Menù

```
localhost:8080/cliente/menu
```

API di Get con la quale si richiede il menù (lista di piatti disponibili)

**Risposta:**

Lista di oggetti piatto:

- `id` : identificativo del piatto
  - `idIngPrinc` : identificativo dell'ingrediente principale
  - `descrizione` : descrizione del piatto
  - `prezzo` : prezzo del piatto
  - `tPreparazione` : tempo medio di preparazione del piatto
- 

### GET Get piatto from menù

```
localhost:8080/cliente/menu/car123
```

API di Get con la quale si richiede uno specifico piatto dal menù

**Parametri della query string:**

- `idpiatto` (obbligatorio): Identificatore del piatto da recuperare

**Risposta:**

- `id` : identificativo del piatto
  - `idIngPrinc` : identificativo dell'ingrediente principale
  - `descrizione` : descrizione del piatto
  - `prezzo` : prezzo del piatto
  - `tPreparazione` : tempo medio di preparazione del piatto
- 

## GET Get new Comanda from Cliente

localhost:8080/cliente/tavolo1/comanda/new

API di Get con la quale si richiede una nuova comanda per un determinato cliente

**Parametri della query:**

- `idCliente` (obbligatorio, stringa) : identificativo del cliente

**Risposta:**

- `id` : identificativo della comanda
  - `idCliente` : identificativo del cliente
  - `codicePagamento` : codice di pagamento
  - `totaleScontrino` : costo totale dei piatti ordinati a scontrino
- 

## GET Get Comanda attiva from Cliente

localhost:8080/cliente/tavolo1/comanda/attiva

API di Get con la quale si richiede la comanda attiva per un dato cliente

**Parametri della query:**

- `idCliente` (obbligatorio, stringa) : identificativo del cliente

**Risposta:**

- `id` : identificativo della comanda
- `idCliente` : identificativo del cliente
- `codicePagamento` : codice di pagamento
- `totaleScontrino` : costo totale dei piatti ordinati a scontrino

## POST Post new Order by Cliente

localhost:8080/cliente/order/add?idcliente=tavolo1&idpiatto=car123&urgenzacliente=0

API di Post con la quale si salva un ordine per un dato cliente all'interno del sistema

### Parametri della query:

- `idcliente` (stringa, obbligatorio) : identificativo del cliente
- `idpiatto` (stringa, obbligatorio) : identificativo del piatto
- `urgenzacliente` (numero, obbligatorio): Attributo urgenza del cliente(0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

### Risposta:

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

## PARAMS

<code>idcliente</code>	tavolo1
<code>idpiatto</code>	car123
<code>urgenzacliente</code>	0

## GET Get order

```
localhost:8080/cliente/order?id=2
```

API di Get con la quale si richiede un ordine specifico

**Parametri della query:**

- `id` (numero, obbligatorio) : identificativo dell'ordine richiesto

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

## PARAMS

---

<code>id</code>	2
-----------------	---

## GET Get Order status

```
localhost:8080/cliente/order/status?id=2
```

API di Get con la quale si richiede lo stato di un ordine specifico

**Parametri della query:**

- `id` (numero, obbligatorio) : identificativo dell'ordine richiesto

**Risposta:**

- `ordine` : Identificatore dell'ordine
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in

preparazione, 3: Ordine preparato)

## PARAMS

---

<b>id</b>	2
-----------	---

## GET Get orders from cliente

localhost:8080/cliente/tavolo1/orders

API di Get con la quale si richiede una lista di tutti gli ordini per un dato cliente

Parametri della query string:

- `idCliente` (obbligatorio, stringa): Identificatore del cliente in questione

Risposta:

Lista di oggetti Ordine:

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

## Gestione Cucina

Il microservizio Gestione Cucina si occupa principalmente di ricevere gli ordini da preparare da Gestione Comanda, disporli nella corretta coda di postazione e gestire l'interazione di queste postazioni.

La comunicazione con gli altri microservizi avviene tramite Message Broker come segue:

- Il microservizio GestioneComanda comunica verso GestioneCucina tramite il topic Kafka SendOrderEvent.
- Il microservizio GestioneCucina comunica verso GestioneComanda tramite il topic Kafka NotifyPrepEvent.

Le API riguardano l'interazione delle postazioni di lavoro con il servizio

---

### GET Get CodaPostazione from Cucina

```
localhost:8080/cucina/codapostazione/riso
```

API di Get con la quale è possibile ottenere la coda della postazione corrispondente all'identificativo di ingrediente principale specificato

**Parametri della query string:**

- `ingredientePrincipale` (obbligatorio): identificativo della coda di postazione

**Risposta:**

- `ingredientePrincipale` : identificativo della coda di postazione
- `numeroOrdiniPresenti` : numero ordini presenti in coda
- `gradoRiempimento` : grado di riempimento attuale della coda
- `capacita` : capacita' massima della coda
- `queue` : coda di ordinazioni (lista di oggetti ordine)

---

### GET Get nextOrder from CodaPostazione

```
localhost:8080/cucina/codapostazione/riso/nextorder
```

API di Get con la quale è possibile ricevere l'ordine che deve essere preparato per una determinata postazione della cucina

**Parametri della query string:**

- `ingredientePrincipale` (obbligatorio): identificativo della coda di postazione

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `idPiatto` : Identificatore del piatto ordinato dal cliente
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)
- `tOrdinazione` : Istante temporale in cui viene effettuata l'ordinazione
- `urgenzaCliente` : Attributo urgenza del cliente ( 0 : espresso non urgenza, 1 : espresso urgenza, 2 : default)

**POST Post NotificaOrdine from cucina**

localhost:8080/cucina/codapostazione/riso

API di Post con la quale è possibile notificare l'avvenuta preparazione di un ordine da parte di una determinata postazione della cucina

**Parametri della richiesta (body JSON):**

- `id` (numero intero, obbligatorio) : Identificatore dell'ordine
- `idComanda` (numero intero, obbligatorio) : Identificatore della comanda di cui l'ordine fa parte
- `stato` (numero intero tra 0 e 3, obbligatorio) : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)

**Risposta:**

- `id` : Identificatore dell'ordine
- `idComanda` : Identificatore della comanda di cui l'ordine fa parte
- `stato` : Stato dell'ordine (0: Ordine preso in carico, 1: Ordine in coda di preparazione, 2: Ordine in preparazione, 3: Ordine preparato)

**Body** raw (json)

json

```
{  
  "id" : 1,  
  "idComanda" : 7,  
  "stato" : 3  
}
```



# Iterazione 2

## 3.1 Algoritmo

### 3.1.1 Briefing

Nell'ambito di questa applicazione si considera che ogni piatto sia composto da un ingrediente principale e da più ingredienti secondari. Ogni piatto ordinato viene chiamato ordine, quindi un ordine comprende un singolo piatto, mentre la comanda contiene tutti gli ordini di un singolo cliente. Nel corso di un brainstorming, si è maturata l'idea di organizzare la cucina in postazioni, ognuna focalizzata su un ingrediente principale: ogni postazione si occuperà quindi di preparare e completare piatti accomunati dallo stesso ingrediente principale.

### 3.1.2 Organizzazione

Di seguito viene illustrata l'organizzazione delle entità coinvolte nella gestione dell'algoritmo:

#### Ordine

Ogni ordine contiene un singolo piatto del menù, viene classificato per ingrediente principale univoco (es. riso, pasta, pesce, . . . ), ogni ordine presenta poi più parametri, questi contribuiscono a calcolare la priorità ad esso associata. Parametri ordine:

- Ingrediente principale;
- Tempo di preparazione;
- Numero ordine effettuato (primo, secondo, . . . );
- Urgenza del cliente;
- Tempo in attesa.

#### Cucina

La cucina viene organizzata in postazioni di lavoro, ossia delle aree dedicate organizzate per svolgere specifiche attività culinarie adibite alla preparazione di piatti che hanno in comune il medesimo ingrediente principale, nello specifico:

- ogni postazione di lavoro è adibita al massimo a 1 ingrediente principale;
- ogni postazione di lavoro può avere più cuochi (la presenza di più cuochi aumenta la velocità di preparazione della postazione), i cuochi possono spostarsi tra le postazioni;
- una postazione può essere vuota, esiste un massimo numero di cuochi per postazione; ogni postazione ha una coda di ordini da preparare:
  - soglia minima di ordini in coda per poter attivare la postazione;
  - soglia massima di ordini in coda (oltre la quale si può richiede un cuoco aggiuntivo oppure di rallentare aggiornando il parametro);
  - tempo massimo in cui gli ordini possono stare in coda di preparazione.
- In preparazione possono stare un numero di ordini pari al numero di cuochi;
- la somma degli ordini in coda di preparazione è sempre minore della lunghezza della coda di preparazione più piccola.

### **Postazione**

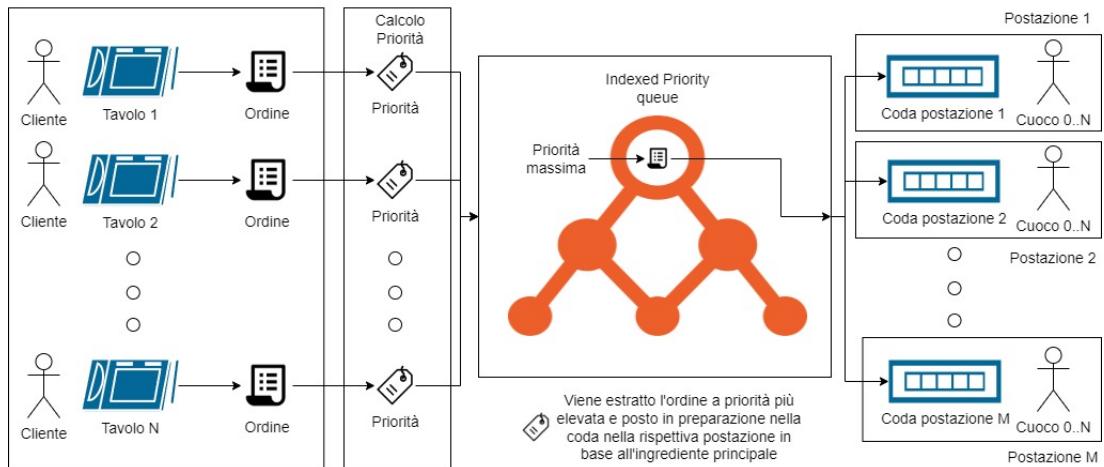
Con postazione si intende uno spazio di lavoro attrezzato con gli strumenti necessari per lavorare con un particolare tipo di ingrediente principale. Una singola postazione presenta una struttura dati per gestire gli ordini in coda di preparazione, ogni postazione presenta un numero massimo di cuochi che possono lavorare contemporaneamente e può essere attivata solo con un numero minimo di ordini in coda (può essere vuota senza cuochi).

- 1 ingrediente principale;
- N cuochi ( $N < M$  max cuochi per postazione);
- 1 struttura dati (coda);
- stato (vuota, regolare, intasata).

### **3.1.3 Struttura dati**

Per quanto riguarda il flusso di un ordine all'interno del sistema si considera che immediatamente dopo l'ordinazione da parte del cliente viene assegnata una priorità per tale ordine, gli ordini vengono così raccolti nella struttura dati principale con l'etichetta della priorità. Successivamente, se la cucina lo richiede, l'ordine con la priorità più elevata viene spostato nella coda di preparazione della rispettiva postazione di lavoro.

Si rendono quindi necessarie due tipi di strutture dati:



**Figura 3.1:** Strutture dati dell'algoritmo

- Struttura dati principale: Indexed priority queue;
- Struttura dati delle postazioni: Coda (queue).

## Indexed priority queue

Struttura dati che estende il concetto di coda con priorità aggiungendo la possibilità di accedere in tempo costante agli elementi presenti in coda per compiere operazioni quali la modifica dei parametri, l'aggiornamento della priorità o la rimozione dell'ordine (che altrimenti presenterebbe costo lineare). Viene implementata per mezzo di una combinazione di una coda con priorità (max heap) e un dizionario (hashtable) che tiene traccia della posizione di ogni elemento all'interno della coda.

**Analisi complessità:** La complessità temporale è correlata a quella di un heap binario, potenziato dall'accesso diretto agli elementi tramite dizionario, di conseguenza:

- creazione:  $O(n)$ ;
- inserimento e rimozione:  $O(\log n)$ ;
- modifica priorità:  $O(\log n)$ ;
- accedere a un elemento:  $O(1)$ .

## Requisiti funzionali:

- Gestione degli ordini con priorità: funzionalità chiave della struttura dati, gli ordini ricevono una priorità prima di entrare nella coda a priorità indicizzata;

- Fornire l'ordine con priorità più elevata: la struttura dati deve essere in grado di fornire alla cucina l'ordine con la priorità più alta quando richiesto;
- Accesso, modifica e rimozione degli ordini: la coda a priorità deve poter fornire la possibilità di implementare la funzionalità che consente ai clienti di accedere, modificare o rimuovere il proprio ordine (nelle prossime iterazioni);
- Flessibilità nella modifica delle priorità: la struttura deve garantire una certa flessibilità alla modifica delle priorità degli ordini, poiché le priorità possono cambiare per conto dei clienti, della cucina e a intervalli regolari di tempo.

**Requisiti non funzionali:**

- Tempo di risposta rapido: l'ordine con priorità più elevata deve essere fornito in tempo rapido alla cucina senza ritardi;
- Tempo di accesso, modifica e rimozione ragionevole: il cliente deve poter effettuare operazioni senza complicazioni in tempi ragionevoli, mantenendo un'esperienza di utilizzo piacevole;
- Scalabilità: La struttura dati deve essere in grado di gestire un grande volume di ordini, adattandosi alle variazioni nella domanda senza compromettere le prestazioni;
- Flessibilità alle modifiche: requisito non funzionale relativo alla flessibilità e alla manutenibilità del sistema.

**Coda (queue)**

Struttura dati lineare che segue il principio "First In, First Out" (FIFO), ossia il principio per il quale il primo elemento che entra nella coda è poi il primo che esce.

**Analisi complessità:**

- inserimento in coda:  $O(1)$ ;
- rimozione della testa:  $O(1)$ ;
- verifica stato:  $O(1)$  se vuota,  $O(n)$  altrimenti.

**Requisiti funzionali:**

- Funzionamento FIFO: La coda deve garantire il corretto funzionamento FIFO (First In, First Out), indipendentemente dalle priorità degli ordini;
- Soglia di attivazione: Il sistema deve permettere di configurare una soglia di valore minimo di attivazione per la coda, al di sotto della quale la postazione non viene attivata;
- Soglia critica di intasamento: Il sistema deve permettere di configurare una soglia di valore critico, oltre la quale la postazione diventa intasata e richiede operazioni per ridurre il carico.

**Requisiti non funzionali:**

- Lunghezza finita della coda: Il sistema deve gestire una coda con una lunghezza finita, limitata dalla capacità della postazione di lavoro;
- Tempo massimo di attesa in coda: Il sistema deve garantire che gli ordini non rimangano in coda di preparazione per troppo tempo prima di essere elaborati;
- Attivazione anticipata della postazione: In casi di eccessivo ritardo nella preparazione degli ordini, il sistema può attivare una postazione di lavoro anche se è al di sotto della soglia minima di attivazione.

**3.1.4 Funzione di priorità**

La funzione di priorità è una funzione matematica che assegna un valore numerico decimale di priorità nell'intervallo tra 0 e 1 basandosi sui parametri specifici di ogni ordine. Il primo passo consiste nel processo di normalizzazione dei parametri, il quale permette di standardizzare i valori in modo che siano compresi tra 0 e 1, in maniera tale da mettere i diversi parametri su una scala comune e uniforme

**Parametri**

**x1 ingrediente principale:** Indica il valore di priorità che presenta l'ingrediente predominante dell'ordine, questo valore è influenzato direttamente dallo stato della postazione di lavoro associata in cucina.

- condizione iniziale ogni ingrediente ha valore 0.5
- se la cucina è satura ridurre il valore (min 0)
- se la cucina è scarica aumentare il valore (max 1)

**x2 tempo di preparazione:** Rappresenta la durata stimata necessaria per preparare un determinato ordine.

- normalizzazione:

$$tp_{\text{norm}} = \frac{tp - tp_{\min}}{tp_{\max} - tp_{\min}}$$

con  $tp$ : tempo di preparazione,

$tp_{\max}$  : tempo di preparazione massimo,

$tp_{\min}$  : tempo di preparazione minimo;

- considerare  $x2 = tp_{\text{norm}}$  per prioritizzare ordini più lunghi,  
oppure  $x2 = 1 - tp_{\text{norm}}$  per prioritizzare ordini più brevi.

**x3 urgenza del cliente:** Consente ai clienti di specificare la tempestività con cui desiderano ricevere il proprio ordine, in particolare i clienti possono chiedere espressamente di avere urgenza, al contrario possono specificare di non avere fretta o non dire nulla e tenere un valore di urgenza di default

- 1 se il cliente ha espresso urgenza;
- 0 se ha espresso di ritardare o fare con calma;
- valore neutro standard 0.5.

**x4 numero ordine effettuato:** Specifica il numero dell'ordine del cliente in ordine temporale, in particolare indica la posizione relativa di un ordine all'interno della sequenza di ordini effettuati.

- il primo ordine effettuato ha priorità maggiore, mentre i successivi hanno priorità decrescente;

- normalizzazione:

$$x4 = \frac{noe - 1}{\max_{noe} - 1}$$

con  $noe$ : numero ordini effettuati,

$\max_{noe}$ : massimo numero ordini effettuabili;

- considerare un valore massimo di ordini (es.5 gli ordini dopo il quinto sono comunque consentiti e prenderanno la stessa priorità del 5° ordine).

**x5 tempo in attesa:** Rappresenta il periodo di tempo trascorso da quando un ordine è stato effettuato fino al momento in cui viene elaborato.

- normalizzazione:

$$x5 = \frac{\text{tempo in attesa}}{\text{tempo max in attesa}}$$

- considerare un valore massimo di tempo in attesa consentito, in prossimità del quale si ha la priorità più elevata.

## Pesi

I pesi sono utilizzati per attribuire un grado di importanza relativo a ciascun parametro all'interno della funzione di priorità. Questi pesi indicano quanto ciascun parametro dovrebbe influenzare il calcolo complessivo della priorità di un determinato elemento. Si elencano di seguito i pesi per ciascun parametro definito poc'anzi:

- p1: peso ingrediente principale;
- p2: peso tempo di preparazione;
- p3: peso urgenza cliente;
- p4: peso numero ordine;
- p5: peso tempo in attesa.

Viene quindi fatto un ragionamento sull'importanza da attribuire a ogni parametro tramite l'incidenza assegnata al singolo peso. Si dividono quindi i pesi in tre categorie.

**Maggiore incidenza** I pesi che devono essere più incidenti sono:

- p1 peso ingrediente principale: per evitare di sovraccaricare una postazione rispetto alle altre o per non avere postazioni vuote;
- p5 peso tempo in attesa: un ordine non può restare in attesa troppo a lungo.

**Incidenza media** Il peso con incidenza media è:

- p3 peso urgenza del cliente: è meno importante dei vincoli di sovraccarico e attesa, ma deve essere comunque una scelta significativa.

**Bassa incidenza** I pesi con bassa incidenza sulla priorità sono:

- p2 peso tempo di preparazione: in confronto ad altri parametri con pesi più elevati, questo è considerato meno critico;
- p4 peso numero ordine effettuato: ha un impatto di poco conto sulla priorità dell'ordine.

### Valore dei pesi

I valori dei pesi vengono quindi definiti inizialmente:

- p1 = 0.25;
- p2 = 0.15;
- p3 = 0.20;
- p4 = 0.15;
- p5 = 0.25.

Questi valori possono essere regolati col tempo per aumentare l'efficienza dell'algoritmo, diventa così importante raccogliere dati storici per poterli analizzare e comprendere come i vari parametri influenzano le prestazioni del sistema, oltre a raccogliere feedback dei clienti, sulla base di ciò sarà richiesto un tuning dei pesi più accurato. Per questo motivo è richiesta una certa flessibilità in modo da consentire l'aggiornamento dei pesi dei parametri in modo dinamico.

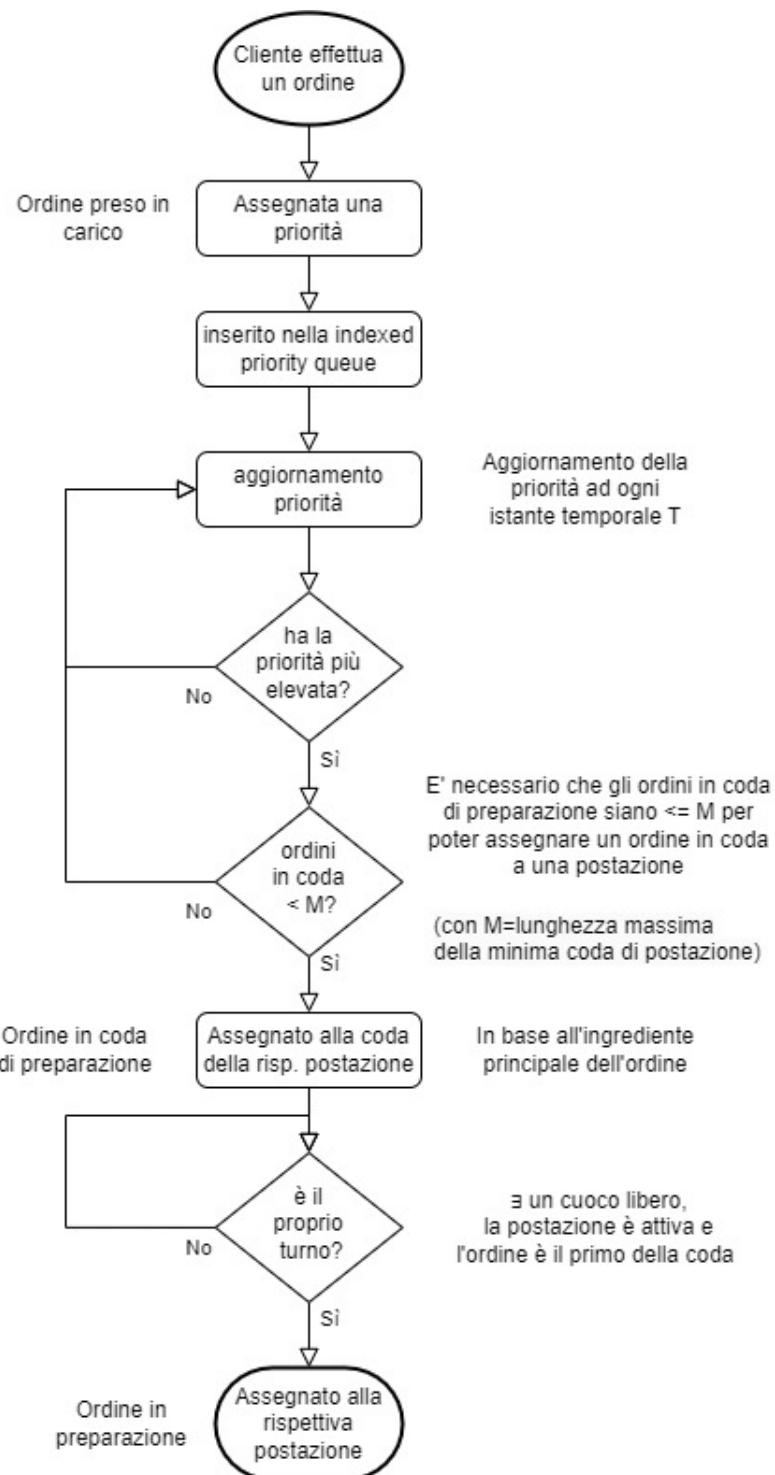
### Funzione matematica

L'equazione proposta rappresenta una somma pesata dei parametri, dove ciascun parametro ( $x_1, x_2, x_3, x_4, x_5$ ) viene moltiplicato per il suo relativo peso ( $p_1, p_2, p_3, p_4, p_5$ ). I pesi indicano l'importanza relativa dei parametri nel determinare la priorità complessiva di un elemento. La somma pesata dei parametri produce un valore ( $y$ ) compreso tra 0 e 1, dove 0 indica un valore meno urgente e 1 indica un valore più urgente.

$$y = p_1 * x_1 + p_2 * x_2 + p_3 * x_3 + p_4 * x_4 + p_5 * x_5$$

### 3.1.5 Diagramma di flusso

Per comprendere meglio il processo dell'algoritmo viene mostrato il diagramma di flusso in Figura 3.2, nel quale viene mostrato il flusso di un ordine dal momento in cui viene effettuato dal cliente a quando viene assegnato alla postazione di lavoro in cucina.

**Figura 3.2:** Diagramma di flusso



# Bibliografia

- [1] Baeldung. *Guide on Loading Initial Data with Spring Boot*. URL: <https://www.baeldung.com/spring-boot-data-sql-and-schema-sql> (visitato il 02/05/2024).
- [2] Baeldung. *Interface Driven Controllers*. URL: <https://www.baeldung.com/spring-interface-driven-controllers> (visitato il 02/05/2024).
- [3] Baeldung. *MockMvc Integration Tests*. URL: <https://www.baeldung.com/integration-testing-in-spring> (visitato il 02/05/2024).
- [4] Baeldung. *Object Mapper*. URL: <https://www.baeldung.com/jackson-object-mapper-tutorial> (visitato il 02/05/2024).
- [5] Baeldung. *Spring Data JPA @Query*. URL: <https://www.baeldung.com/spring-data-jpa-query> (visitato il 02/05/2024).
- [6] Baeldung. *Testing Kafka*. URL: <https://www.baeldung.com/spring-boot-kafka-testing> (visitato il 02/05/2024).
- [7] Happy Coders. *Filosofia Esagonale e Microservizi*. URL: <https://www.happycoders.eu/software-craftsmanship/hexagonal-architecture/> (visitato il 02/05/2024).
- [8] Continuous Integration with GitHub Actions. URL: <https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration> (visitato il 02/05/2024).
- [9] Roman Glushach. *Hexagonal Architecture: The Secret to Scalable and Maintainable Code for Modern Software*. URL: <https://romanglushach.medium.com/hexagonal-architecture-the-secret-to-scalable-and-maintainable-code-for-modern-software-d345fdb47347> (visitato il 02/05/2024).
- [10] H2 DB. URL: [https://www.h2database.com/html/features.html#in-memory\\_databases](https://www.h2database.com/html/features.html#in-memory_databases) (visitato il 02/05/2024).
- [11] Arho Huttunen. *Testing Jackson*. URL: <https://www.arhohuttunen.com/spring-boot-json-test/> (visitato il 02/05/2024).
- [12] Kevin. *How to Implement Port and Adapters in Hexagonal Architecture with Java*. URL: <https://1kevinson.com/how-to-implement-port-and-adapters-in-hexagonal-architecture-with-java/> (visitato il 02/05/2024).
- [13] Spring Boot. URL: <https://spring.io/projects/spring-boot> (visitato il 02/05/2024).

- [14] *Spring Kafka*. URL: <https://docs.spring.io/spring-kafka/reference/index.html> (visitato il 02/05/2024).
- [15] *Web Layer Test*. URL: <https://spring.io/guides/gs/testing-web> (visitato il 02/05/2024).