Università degli Studi di Salerno

Corso di Ingegneria del Software

RistOrganizer
System Design Document
Versione 1.1

Progetto: RistOrganizer	Versione: 1.0
Documento: System Design Document	Data: 20/11/2024

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	**Matricola**
Stefano Palazzo	0512117736
Francesco Casillo	0512118276
Giulia Troiano	0512118888
Vincenzo Ferrara	0512117970

Scritto da:	Stefano Palazzo

Revision History

Data	**Versione**	**Descrizione**	**Autore**
20/11/2024	1.0	Prima Versione	Stefano Palazzo
25/11/2024	1.1	Aggiunto Subsystems services	Stefano Palazzo

Indice

1 Introduzione

1.1 Scopo del sistema

Il sistema si propone di fornire gli strumenti per la gestione delle prenotazioni e dell'organizzazione di un locale di ristoro.

Gli obiettivi principali del sistema sono quelli di agevolare la prenotazione da parte del cliente attraverso un sistema di prenotazioni online, facilitare e velocizzare la comunicazione tra i componenti del personale ed infine sfruttando le tecnologie web fornire all'amministratore gli strumenti necessari alla gestione dei componenti del sistema.

1.2 Obiettivi del Design

a. **Throughput**: Il sistema è utilizzabile su più dispositivi contemporaneamente.
b. **Usabilità**: Il sistema verrà utilizzato da diversi tipi di utenti, ognuno accederà alla propria area di interesse, potrà accedere ad informazioni specifiche ed effettuare determinate operazioni.
c. **Affidabilità**: il sistema deve essere affidabile. I crash del sistema verranno gestiti in modo da non compromettere le sue funzionalità. Il sistema è basato sul web e su un server di database relazionale.
d. **Efficienza**: Il sistema svolgerà i propri compiti in tempi accettabili utilizzando nel miglior modo le risorse disponibili.
e. **Robustezza**: il software deve essere facilmente modificabile a fronte di ogni evenienza.
f. **Funzionalità**: il sistema deve soddisfare le necessità del cliente evidenziate in fase di raccolta e analisi dei requisiti.
g. **Adattamento**: il software deve essere modificabile in base alle esigenze del cliente o per una futura revisione.
h. **Sicurezza**: Il sistema deve essere immune ad accessi non autorizzati e solo gli utenti autorizzati devono poter accedere alla loro area di interesse.

1.3 Definizioni e acronimi

- **Acronimi**

SDD: System Design Document

RAD: Requirements Analysis Document

- **Definizioni**

Web-Browser: software che permette la navigazione sul web.

Web server: server che permette agli utenti di visualizzare pagine web attraverso il web-browser.

Database: struttura dati che consente di immagazzinare dati sul web.

1.4 Riferimenti

- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (3nd edition), Prentice- Hall, 2003.
- Jim Arlow, Ila Neustadt, UML e Unified Process, McGraw-Hill Italia Ian Sommerville,I. Sommerville, Software Engineering (9th edition), Addison Wesely.

2 Architettura di software corrente

Il sistema è stato implementato dal nulla, in quanto non esiste alcun sistema simile al nostro.

La raccolta dei requisiti viene effettuata tramite colloqui con vari gestori di locali. Ascoltando le loro esigenze, abbiamo creato un sistema automatizzato ex-novo che sostituisca a pieno l'antiquato sistema di comande cartacee senza il bisogno di apparecchiature eccessivamente ingombranti sia dal punto di vista economico e sia dal punto di vista fisico.

3 Architettura proposta

3.1 Overview

Per la suddivisione in sottosistemi è stata abbandonata la precedente impostazione (mostrata nel RAD) per adottare un pattern di progettazione chiamato Facade Design Pattern che prevede all'interno di ogni sottosistema una classe facade che permette loro di comunicare con i sottosistemi esterni e si occupa di smistare correttamente i messaggi destinati ad ogni model element del sottosistema in cui si trova.

Decomposizioni in sottosistemi

Il sistema è stato dunque suddiviso in diversi sottosistemi:

- DBMS Special Purpose (DBMS_SP)

Si occupa di gestire tutte le richieste al database, le altri classi per comunicare col database dovranno passare attraverso questa classe. La comunicazione è garantita grazie alla classe facade contenuta all'interno del sottosistema. Essa raggiungerà le altre classi utilizzando opportune interfacce esterne ai sottosistemi.

- Utenti

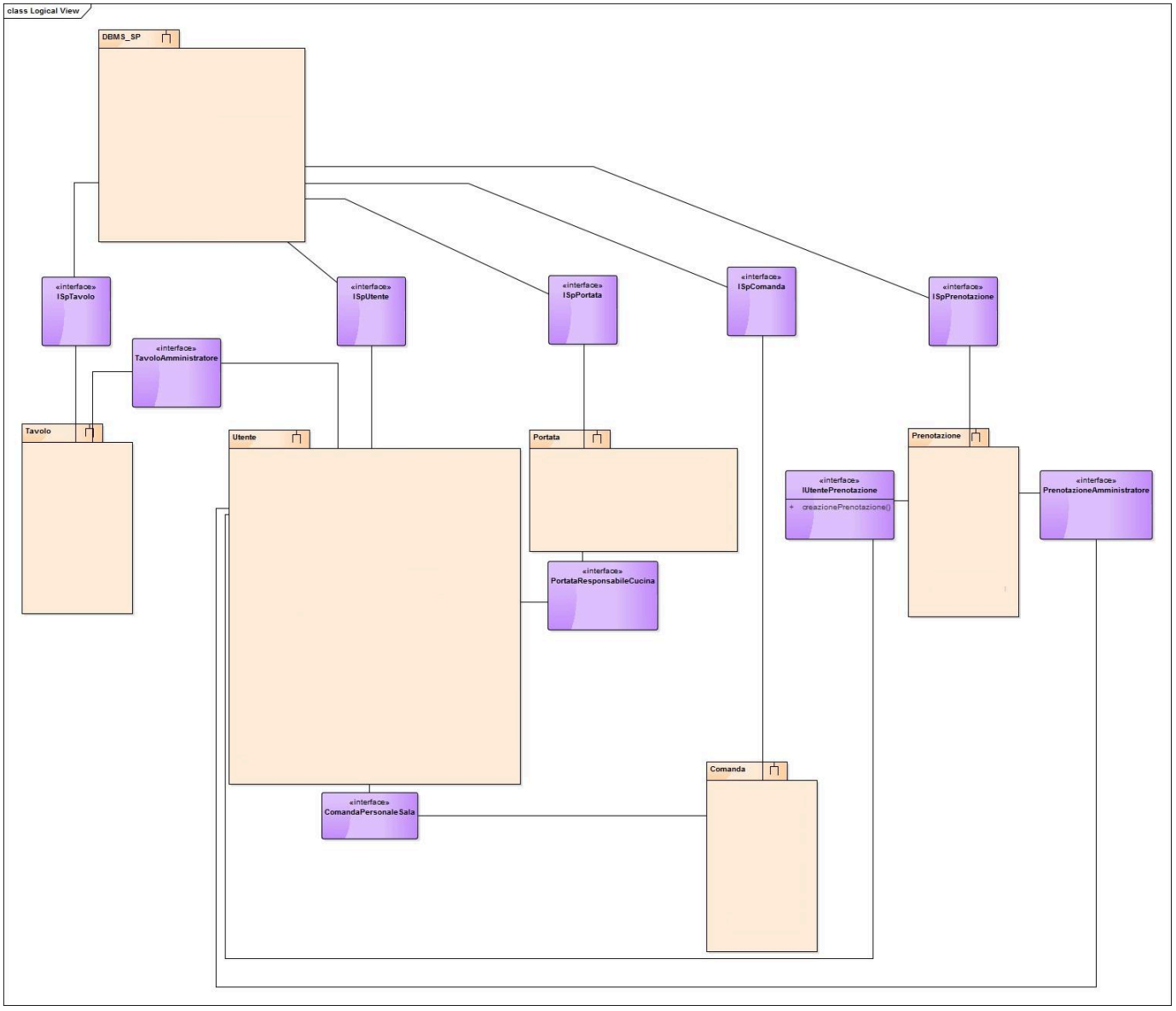
Comprende l'insieme di tutti i tipi di utenti che il nostro sistema gestisce specializzandoli poi quando opportuno. Il sottosistema comunica con l'esterno tramite la classe facade. Le classi specializzate comunicheranno con i sottosistemi attraverso la classe facade e le interfacce (sempre esterne ai sottosistemi)

- Tavolo, Portata, Prenotazione, Comanda

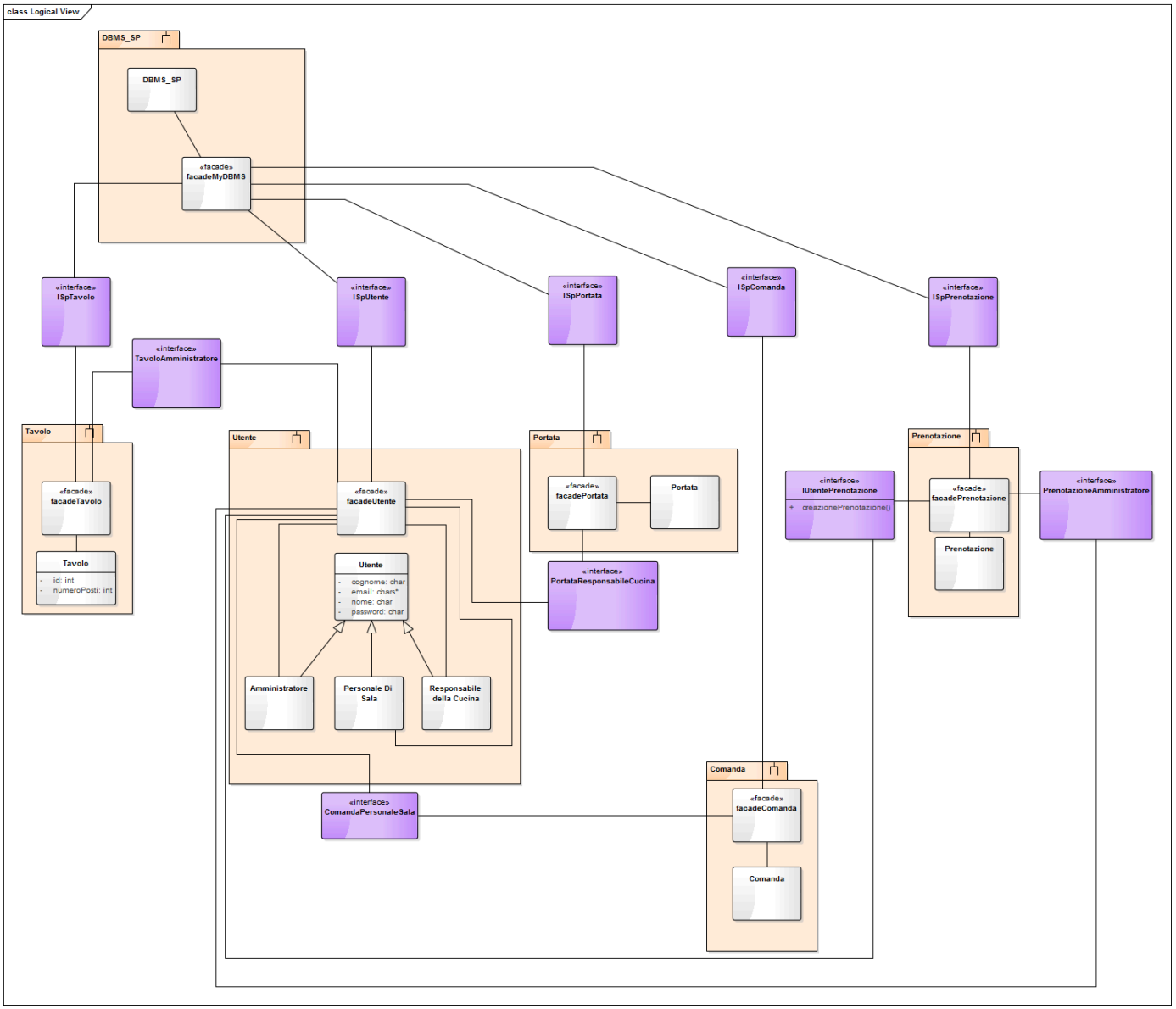
Identici tra di loro come struttura (ma diversi nello scopo), questi sottosistemi hanno al loro interno una classe di smistamento facade che si occuperà delle operazione di I/O di quel sottosistema.

La comunicazione con i sottosistemi esterni avverrà sempre e comunque tramite le interfacce esterne.

Logic Layer – Vista esterna



Logic Layer – Vista interna



3.2 Mapping Hardware/Software

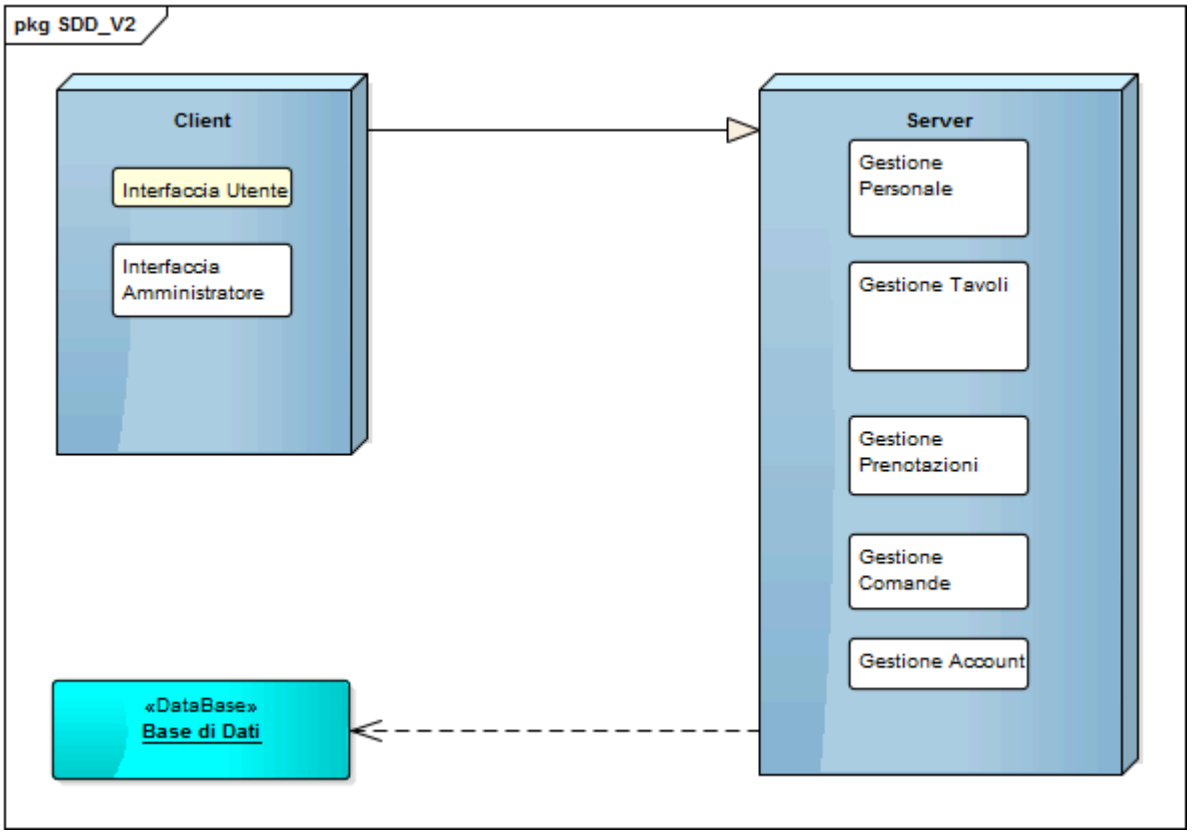
Il sistema si basa su un’architettura Web-based.

Protocollo richiesto: HTTP

Memorizzazione dati: Database MySQL

Web Server: Apache

Linguaggi di programmazione utilizzati: PHP, HTML, CSS, JavaScript



3.3 Gestione dei dati persistenti

I dati persistenti vengono inserito in un opportuno database

- Le credenziali di accesso per amministratore e personale saranno conservate nel database e le password di accesso verranno opportunamente criptate in modo da garantire la sicurezza dei dati.

- Le prenotazioni saranno salvate nel database in modo da poterle gestire tramite il nostro sistema.

- Le ordinazioni/comande verranno inserite nel database in modo da essere gestite dal personale del locale tramite il nostro sistema.

3.4 Controllo d’accesso

Il sistema fornirà un sistema di controllo degli accessi attraverso un meccanismo di login. Ogni tipo di utente potrà accedere esclusivamente alla propria area di competenza del sistema:

- L’amministratore accede al suo pannello dal quale può sfruttare TUTTE le sue funzionalità.

- Il cliente potrà accedere alla homepage del sistema e potrà esclusivamente effettuare prenotazioni e visualizzare informazioni relative al locale.

- Il responsabile di cucina avrà accesso alla gestione delle ordinazioni.

- Il personale di sala avrà accesso al pannello di inserimento delle comande e potrà visualizzare le ordinazioni contrassegnate come “preparate” dal responsabile di cucina.

Attori	Amministratore	Cliente	Responsabile di Cucina	Personale di Sala
Creazione Prenotazioni	x	x		
Gestione Comande	x			x
Gestione Ordinazioni	x		x	x
Gestione Tavoli	x			
Gestione Account	x	x		

3.5 Controllo globale del software

Il sistema RistOrganizer ha un'architettura modulare che garantisce il corretto funzionamento. Sincronizzazione e prestazioni Manutenzione e capacità di scalare in base alle esigenze del sistema. Descrizione delle principali caratteristiche di missione del controllo globale. Strategie e soluzioni visibili agli occhi degli altri Facilitazione e facilità d'uso

Scalabilità

Il sistema è progettato per essere scalabile orizzontalmente o verticalmente:

- **Scalabilità verticale**: È possibile aumentare le risorse hardware del server (CPU, RAM, storage) per supporto e modificare l'architettura del software.
- **Scalabilità orizzontale**: Il sistema può essere distribuito su più server utilizzando tecniche e garantisce la continuità operativa.
- **Cache**: Introdurre meccanismi di caching (ad esempio, radis o memcaching) per database temporanei ed effimeri.
- **Architettura stateless**: Le richieste utente vengono trattate come indipendenti, consentendo l'aggiunta di server senza riconfigurazioni complesse.

Prestazioni

Per garantire un'esperienza utente fluida, il sistema è progettato con obiettivi prestazionali chiari:

- **Tempi di risposta**: Il sistema deve rispondere alla maggior parte delle richieste in tempi brevi, anche durante picchi di carico.
- **Throughput**: Supporto per richieste simultanee con tempi di risposta ottimali.
- **Ottimizzazioni**: Le query SQL sono ottimizzate per minimizzare i tempi di elaborazione.
- **Monitoraggio**: Il sistema include strumenti di monitoraggio per analizzare tempi di risposta e identificare colli di bottiglia.

Manutenibilità

La progettazione del sistema facilita interventi di manutenzione e aggiornamenti futuri grazie alle seguenti caratteristiche:

- **Modularità**: Ogni sottosistema (ad esempio, DBMS_SP, Utenti, Tavolo) è progettato in modo indipendente, consentendo modifiche localizzate senza impatti sugli altri componenti.
- **Documentazione**: Ogni modulo è accompagnato da documentazione dettagliata per agevolare gli sviluppatori nella comprensione e nella manutenzione.

- **Versionamento**: Il sistema utilizza un controllo di versione (es. Git) per gestire gli aggiornamenti in modo strutturato.
- **Testing continuo**: Attraverso pipeline CI/CD, ogni modifica è sottoposta a test automatici per evitare regressioni.
- **Interfacce standardizzate**: Le API sono progettate secondo standard consolidati per facilitare l'integrazione e il debugging.

3.6 Boundary conditions

- **Installazione**:

  Lato server è richiesta la presenza di un database, che verrà configurata attraverso opportuni script inglobati nel sistema. I file necessari all'esecuzione del nostro sistema verranno caricati sul server attraverso una connessione FTP, utilizzando il client FILEZILLA.

- **Avvio e Spegnimento**:

  Il web-server è costantemente in esecuzione, quindi dopo l'installazione verrà praticato il primo avvio (e configurazione) e non sarà quindi necessario riavviare o spegnere il sistema. In caso di interventi di manutenzione il sistema provvederà a limitare l'accesso alle pagine pubbliche mostrando agli utenti una pagina provvisoria.

- **Gestione malfunzionamenti**:

  In caso di blocco del sistema sarà sufficiente riavviare la macchina che ospita il sistema. Il sistema sarà configurato (tramite gli script di configurazione) ad avviarsi all'avvio e per riprendere automaticamente le sue funzioni.

4. Subsystems services

1. Amministratore

- `createReservation(customerId, tableId, dateTime)`
  Crea una prenotazione per un cliente specifico.

- `updateReservation(reservationId, newDetails)`
  Modifica i dettagli di una prenotazione esistente.
- `deleteReservation(reservationId)`
  Elimina una prenotazione specifica.
- `manageTable(tableId, action, details)`
  Gestisce la configurazione dei tavoli (aggiunta, modifica, eliminazione).
- `createUser(username, password, role)`
  Crea un nuovo utente nel sistema con un ruolo specifico.
- `updateUser(userId, updatedDetails)`
  Aggiorna i dettagli di un account utente.
- `deleteUser(userId)`
  Elimina un utente dal sistema.

2. Cliente

- `bookTable(customerId, tableId, dateTime, partySize)`
  Permette al cliente di prenotare un tavolo per un determinato orario.
- `viewReservation(customerId)`
  Mostra al cliente tutte le sue prenotazioni attive.
- `cancelReservation(reservationId)`
  Cancella una prenotazione specifica.
- `viewMenu()`
  Mostra il menu del ristorante disponibile per il cliente.

3. Responsabile di Cucina

- `viewPendingOrders()`
  Visualizza tutte le comande in attesa di preparazione.
- `markOrderAsPrepared(orderId)`
  Segna un'ordinazione come completata.
- `viewOrderDetails(orderId)`
  Visualizza i dettagli di una comanda specifica.

4. Personale di Sala

- `addOrder(tableId, items)`
  Inserisce una nuova comanda per un tavolo specifico.
- `viewPreparedOrders()`
  Visualizza le ordinazioni che sono state completate dalla cucina.

- `confirmOrderDelivery(orderId)`
  Segna un'ordinazione come consegnata al cliente.
- `manageReservation(reservationId, action)`
  Modifica o aggiorna una prenotazione in base alle richieste del cliente.