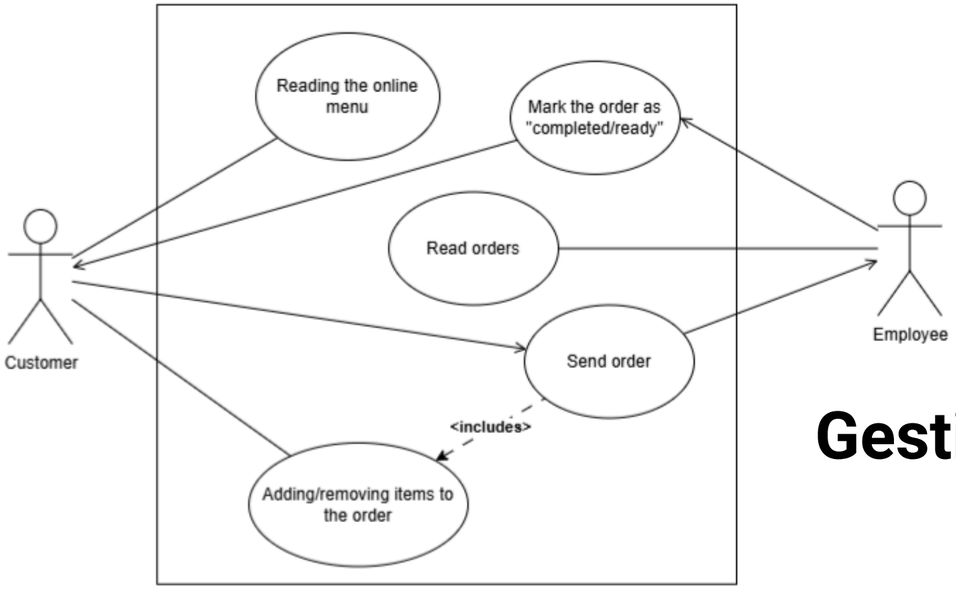
Distributed Cafè

Distributed Systems A.A. 2023/2024

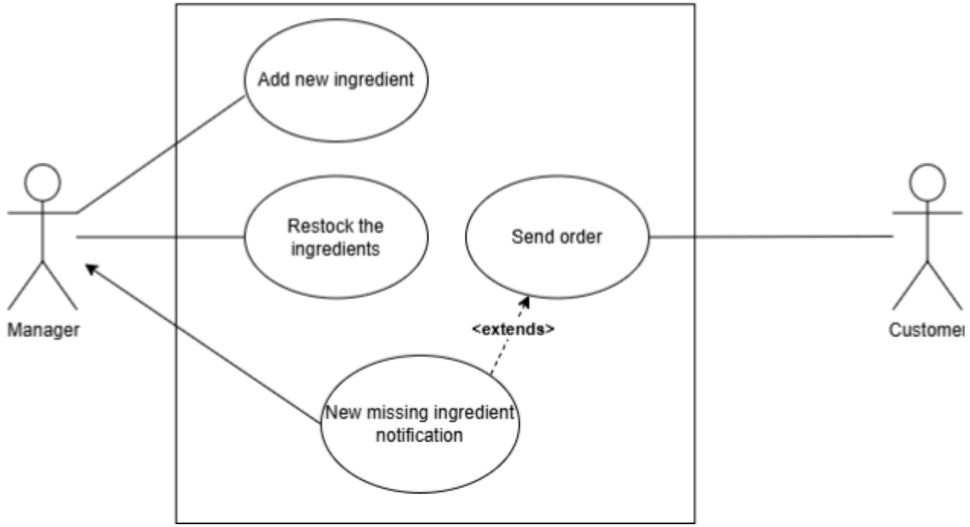
Elisa Albertini e Eleonora Bertoni

Requisiti del progetto - casi d’uso

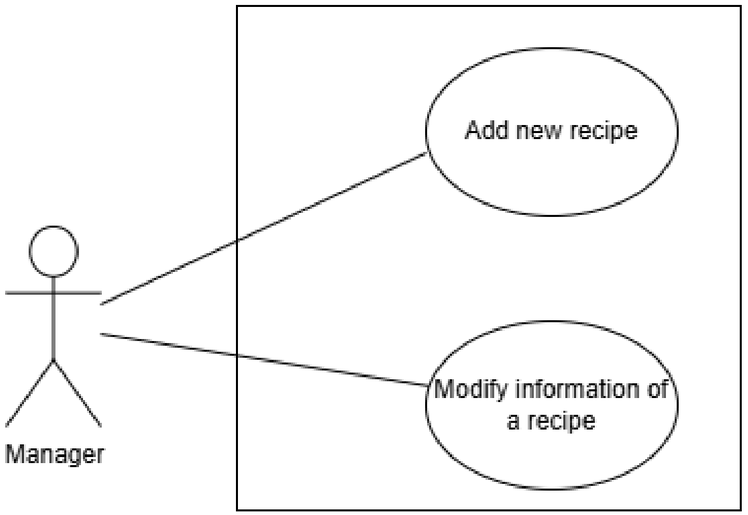
Gestione degli ordini



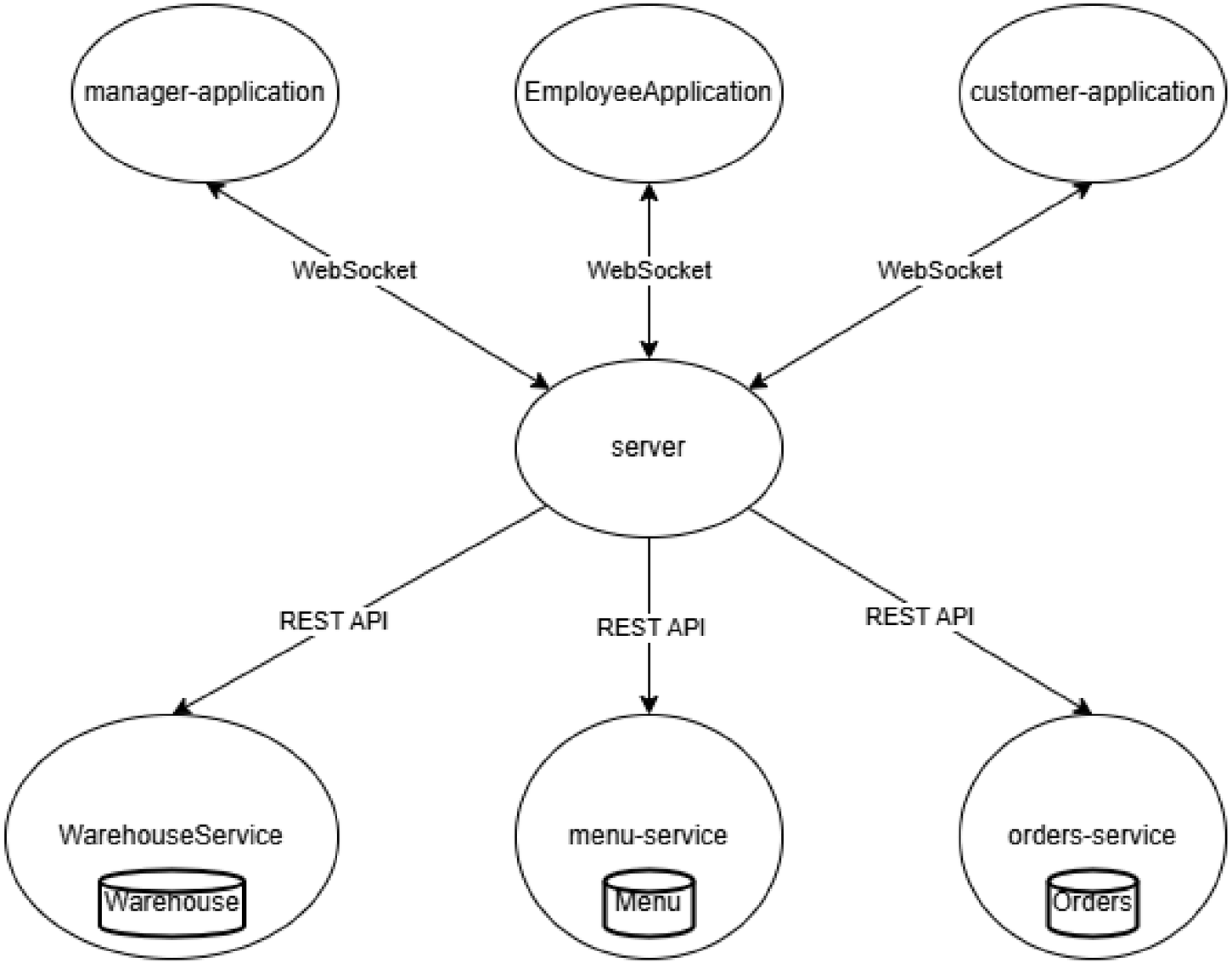
Gestione del magazzino



Gestione del menù

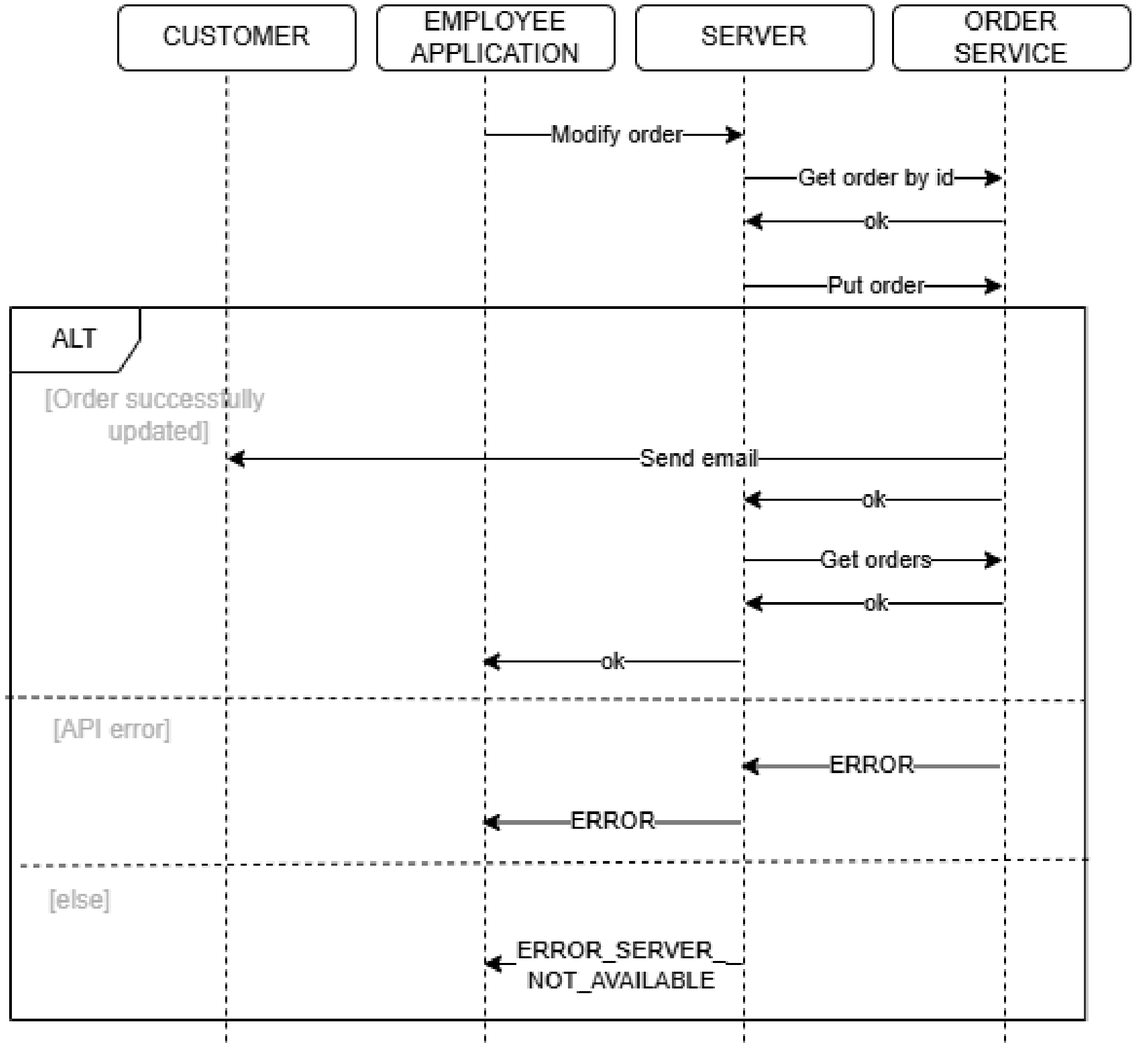


Struttura del sistema distribuito



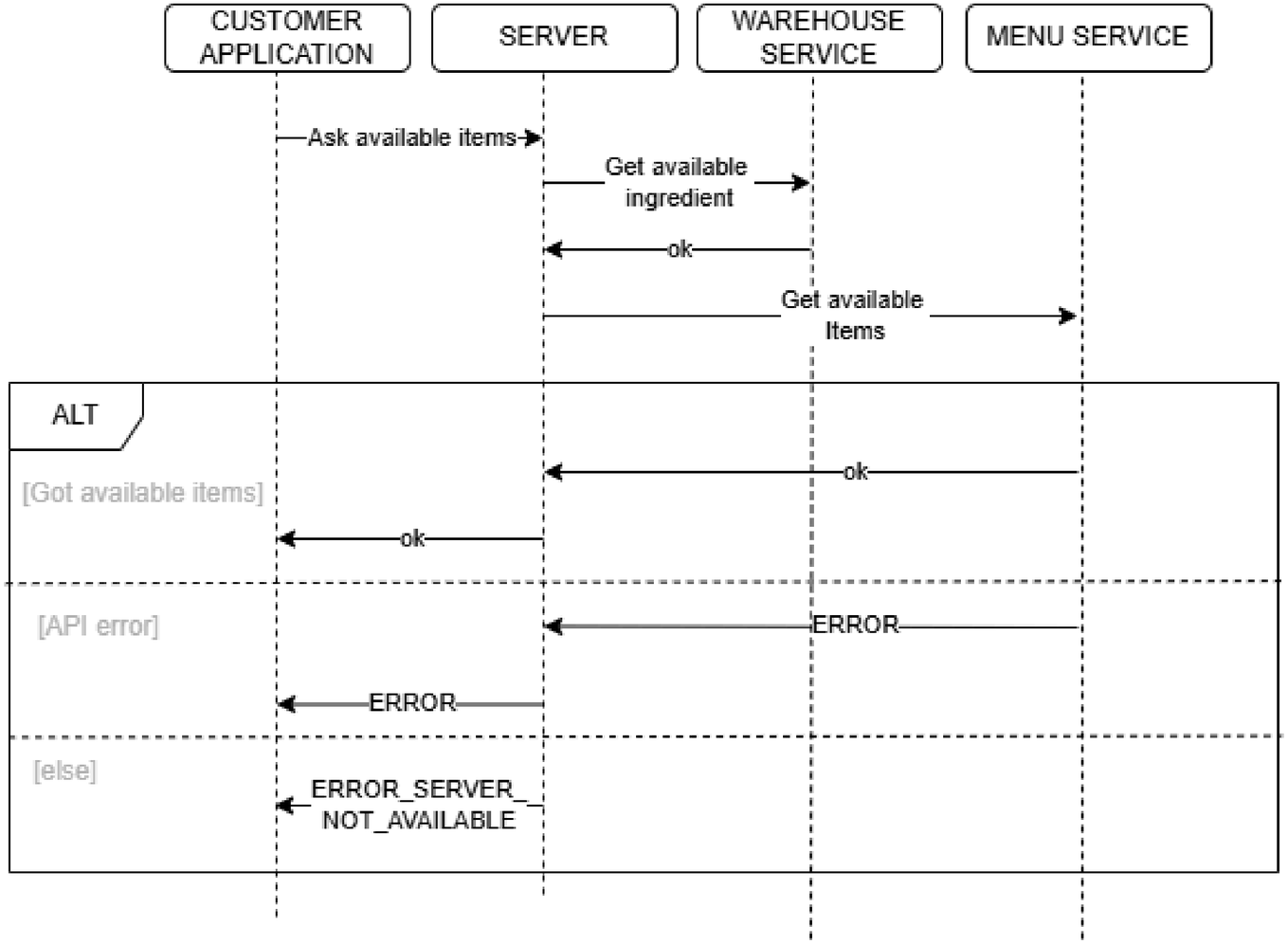
Interazioni

Diagramma di sequenza per la funzionalità “put order”



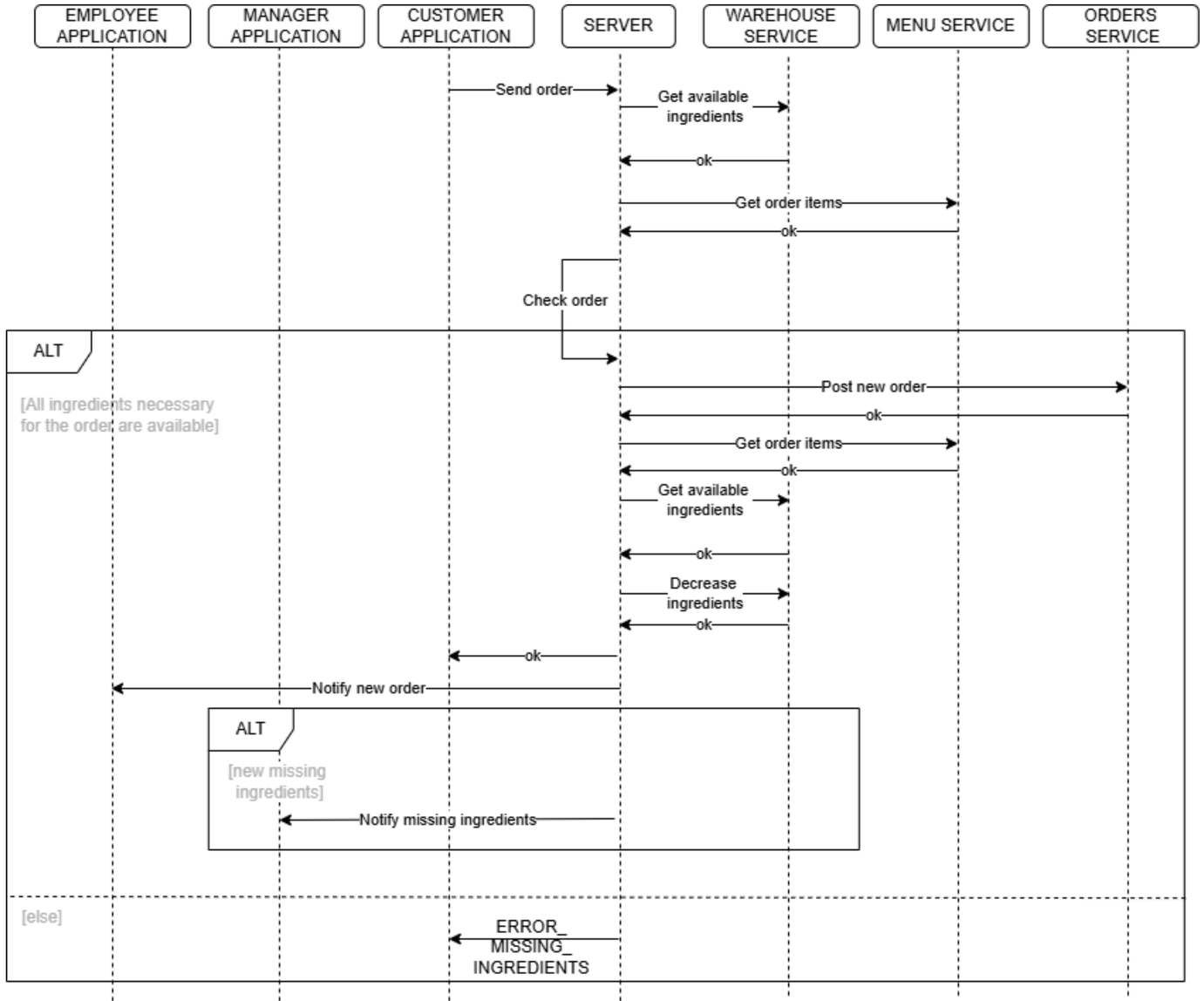
Interazioni

Diagramma di sequenza per la funzionalità “get available items”



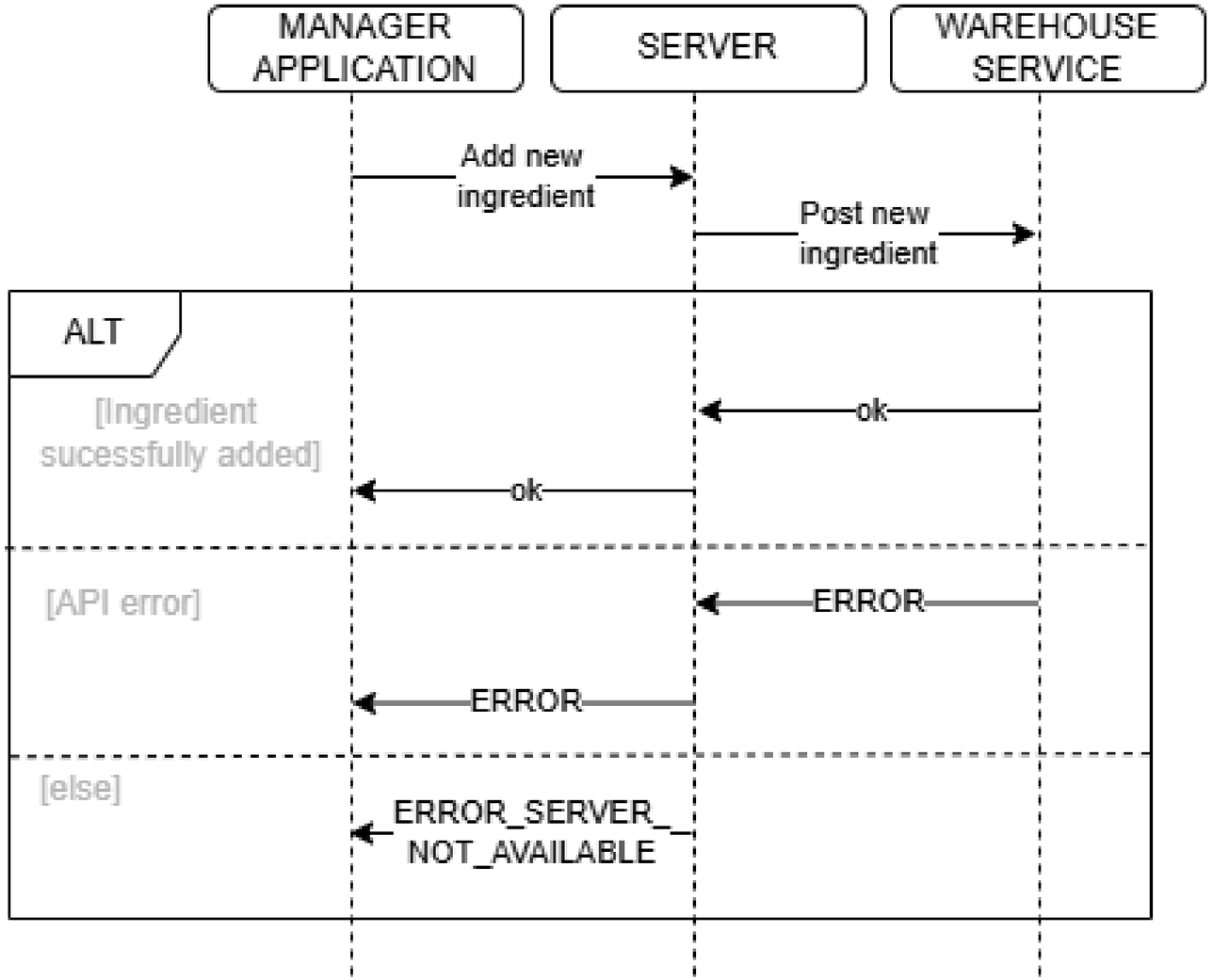
Interazioni

Diagramma di sequenza per la funzionalità “create order”



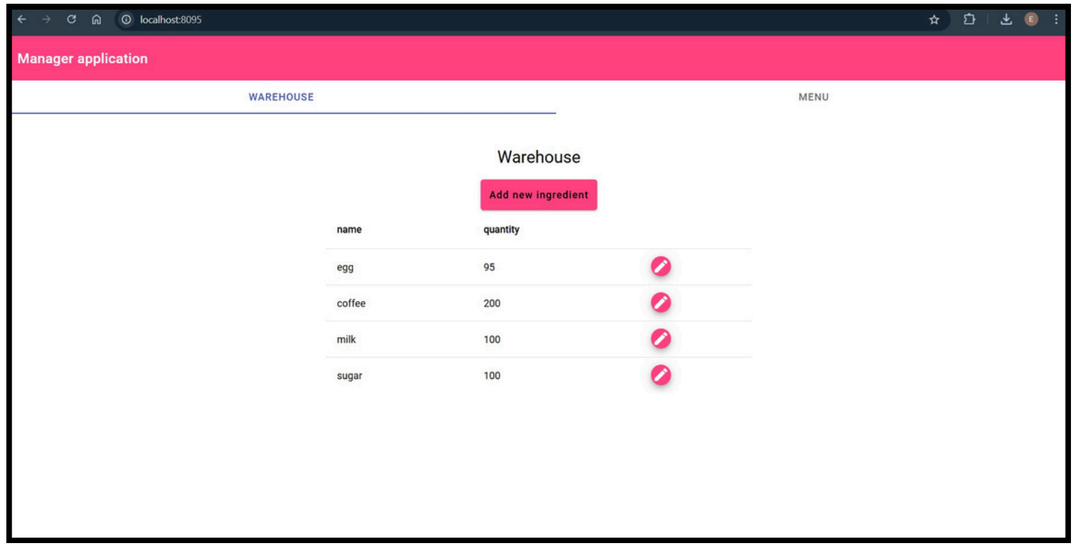
Interazioni

Diagramma di sequenza per la funzionalità “create ingredient”



Architettura front-end

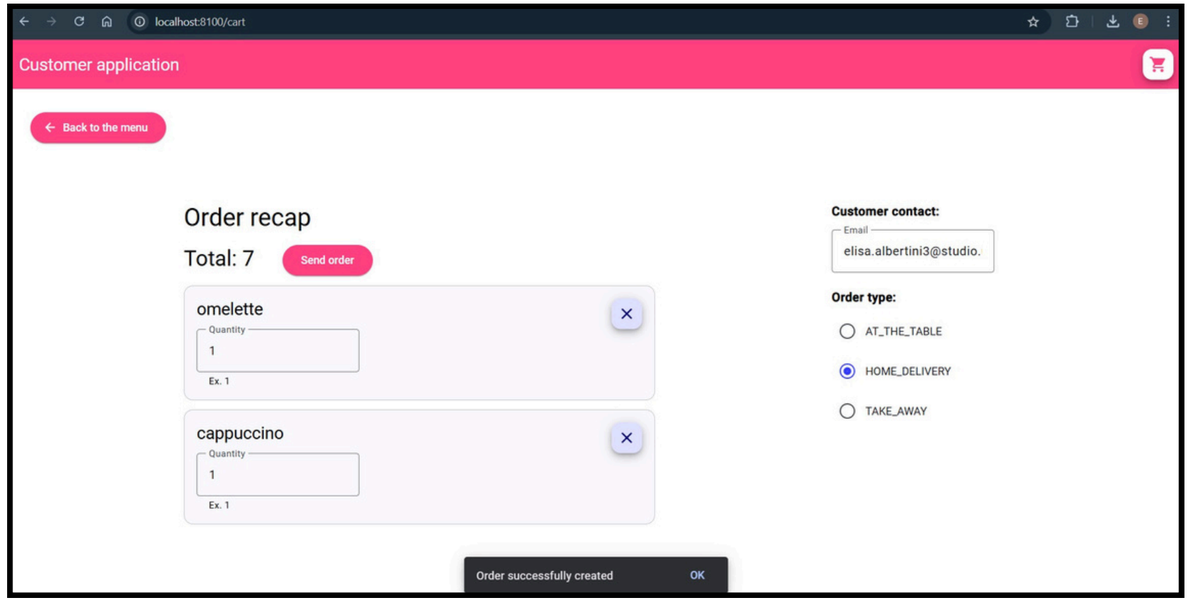
manager-application



- Applicazione Angular
- ***home*** web page
  - utilizzo connessione web socket + Log
  - pop-up per ingredienti esauriti
- componente ***data-table***
  - visualizzazione del contenuto del magazzino/menu
  - utilizzo connessione web socket
  - apertura della dialog che permette di richiedere i dati per aggiornarsi
- cartella ***schema***
  - con le interfacce corrispondenti alla struttura dei JSON

Architettura front-end

customer-application



- Applicazione Angular
- componente ***menu***
  - visualizzazione degli elementi disponibili
  - utilizzo connessione web socket
  - aggiunta di elementi nel carrello
- ***cart*** web page
  - visualizzazione del riepilogo degli elementi del menù selezionati dal cliente
  - utilizzo connessione web socket
  - invio nuovo ordine
- ***cart-storage***
  - implementazione dell'accesso al *local storage*
- cartella ***schema***
  - con le interfacce corrispondenti alla struttura dei JSON

Architettura front-end

EmployeeApplication

- Swing GUI in Java
- classe ***WebsocketConnection***
  - utilizzo connessione web socket + Log
  - aggiornamento dopo notifica
  - riconnessione tramite polling
- classi ***View*** and ***OrderCard***
  - funzionalità di *repaint*
- package ***schema***
  - con le interfacce corrispondenti alla struttura dei JSON

Architettura back-end - microservizi

WarehouseService, orders-service and menu-service

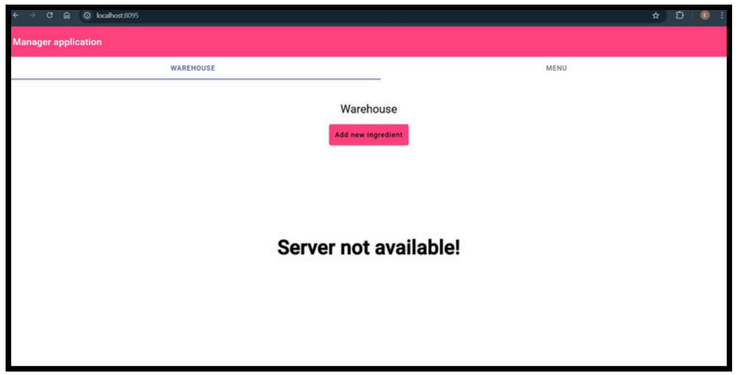
- REST API
- L’architettura è strutturata a layer
  - ***domain***: implementazione delle entità di dominio
  - ***repository***: incapsulamento dell’accesso e della gestione del database
  - ***application***: incapsulamento della business logic
  - ***handler***: gestione delle richieste http
  - ***API***: esposizione delle rotte

Architettura back-end

Server

- ***server.ts***
  - creazione della web socket lato server che ascolta attendendo nuove richieste dal front-end
  - ricezione Log e RequestMessage
- ***check-service.ts***
  - mappa le RequestMessage che arrivano in chiamate alle API e gestendoli mediante handler
- ***handlers.ts***
  - implementazione dei meccanismi di programmazione asincrona
  - gestione dei microservizi inattivi
  - notifica gli altri front-end
- cartella ***schema***
  - con la struttura JSON dei dati e dei messaggi

Architettura - corner cases



- Ogni ***microservizio*** gestisce:
  - tutti gli errori relativi al database
  - tutti gli errori relativi alla business logic
  - input sbagliati:
    - il formato differisce da quello richiesto dalle API
    - alcuni valori potrebbero essere sbagliati per i vincoli del dominio
- Il ***server*** gestisce gli errori dei microservizi ogni volta che un'API viene chiamata, completando l'operazione con un errore appena una delle API fallisce o non può essere raggiunta poiché il microservizio non è disponibile
- ***Front-end***:
  - uso delle risposte date dal server
    - per aggiornare le informazioni visualizzate
    - per visualizzare i messaggi di errore agli utenti

Esempio:

- Errore inviato dal server al front-end quando non è possibile soddisfare una richiesta in quanto il microservizio coinvolto non è disponibile
  - ERROR_SERVER_NOT_AVAILABLE
- Messaggio usato dal microservizio del magazzino per segnalare un errore a livello del database
  - ERROR_EMPTY_WAREHOUSE

Dettagli implementativi

- Technologies
  - Express
  - Axios
  - Vert.x
  - MongoDB
  - Angular
  - Web socket
  - Typia
  - Docker

Documentation:

- Swagger
  - documentazione REST API
  - documentazione delle web socket
- Codice

Docker:

- uso di Docker per fare la costruzione delle immagini di tutti i componenti del sistema, ad esclusione di EmployeeApplication
- uso di Docker Compose per eseguire i container
  - Docker Compose usa le immagini dei container pubblicate

Validazione

Vengono forniti ***Unit test*** per i componenti del *back-end* in modo da validare il corretto funzionamento del nostro sistema

Per ogni ***microservizio*** testiamo:

- REST API
- implementazione

Per il ***server*** testiamo:

- Web socket API
- implementazione
- il corretto invio delle notifiche