Progetto ingegneria del software
A cena con estranei

...

17 settembre 2025

Consegna

Progetto 15 – A cena con estranei

Numero massimo di membri: 4

Descrizione generale:
Si vuole costruire una applicazione simile alla attuale applicazione Tablo che però permetta di organizzare delle cene in casa propria invitando amici o persone estranee.
La cena viene pubblicata sulla piattaforma ed è possibile agli utenti della piattaforma di iscriversi alla cena.
L'oste può rifiutare uno specifico utente che vuole partecipare ad una cena.
L'oste deve descrivere il menù della cena, o del pasto, la località ed altri dettagli.
Commensali e osti ricevono delle recensioni che sono pubbliche.

Requisiti minimi:
- Una Docker per il lato server

Requisiti premiali:
- Docker per i lato client per simulare diversi dispositivi

Fasi principali per la progettazione e lo sviluppo di un software:

Specifica	Sviluppo	Validazione e Evoluzione
• Contiene la raccolta e analisi dei requisiti	• Progettazione • Implementazione	• Testing • Manutenzione

Per la costruire dei modelli di analisi, progetto e implementazione del sistema software è stato utilizzato UML (Unified Model Language) con le diverse viste

Raccolta e analisi dei requisiti

In questa fase l'obiettivo è capire cosa deve fare il sistema, definendo i requisiti e le interazioni principali con gli utenti (senza entrare troppo nel tecnico).

Diagrammi:

- Diagramma dei casi d'uso
- Diagramma di attività

Diagramma dei casi d’uso

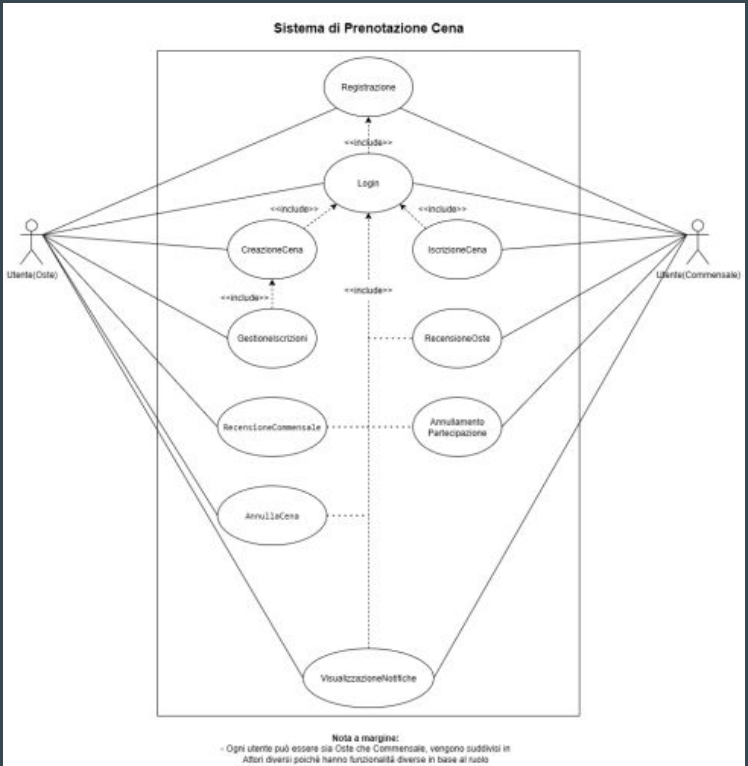


Figure 1: Diagramma UML dei casi d’uso del sistema di prenotazione cena

Modella le funzionalità principali del sistema dal punto di vista dell'utente, mostrando le interazioni tra gli attori e il sistema stesso.

- Attori: chi o cosa interagisce con il sistema
- Casi d’uso: le funzionalità che il sistema deve fornire

Diagramma dei casi d'uso

Registrazione

Use case	Registrazione
ID	1
Brief description	Permette all'utente di creare un account per accedere al sistema.
Primary actors	Oste, Commensale
Secondary actors	-
Preconditions	1. L'utente non è già registrato
Main flow	1. L'utente accede alla schermata di registrazione 2. Inserisce le informazioni richieste (nome, email, password, ecc.) 3. Il sistema verifica i dati 4. Il sistema crea un nuovo account utente
Postconditions	1. L'utente è registrato e può effettuare il login
Alternative flows	1. Se l'email è già in uso, il sistema mostra un messaggio di errore

Login

Use case	Login
ID	2
Brief description	Consente all'utente registrato di accedere alle funzionalità del sistema.
Primary actors	Oste, Commensale
Secondary actors	-
Preconditions	1. L'utente deve essere registrato
Main flow	1. L'utente inserisce credenziali (email, password) 2. Il sistema verifica la correttezza delle credenziali 3. L'accesso viene garantito se le credenziali sono valide
Postconditions	1. L'utente accede al sistema come Oste o Commensale
Alternative flows	1. Se le credenziali sono errate, il sistema blocca l'accesso e mostra un errore

Diagramma dei casi d'uso

Creazione Cena

Use case	CreazioneCena
ID	3
Brief description	L'Oste crea un nuovo evento cena specificando titolo, data, menu e luogo.
Primary actors	Oste
Secondary actors	-
Preconditions	1. L'oste è autenticato nel sistema
Main flow	1. L'oste seleziona l'opzione "Crea Cena" 2. Inserisce i dettagli richiesti: data, orario, descrizione, menù, località 3. Il sistema salva la cena e la pubblica
Postconditions	1. La cena è visibile ai commensali registrati
Alternative flows	1. Se i campi obbligatori non sono compilati, il sistema segnala gli errori

Gestione Iscrizioni

Use case	GestioneIscrizioni
ID	4
Brief description	Permette all'oste di visualizzare e approvare o rifiutare le richieste di partecipazione alla propria cena.
Primary actors	Oste
Secondary actors	-
Preconditions	1. La cena è stata creata
Main flow	1. L'oste visualizza le richieste di iscrizione 2. Per ogni richiesta, può accettare o rifiutare 3. Il sistema aggiorna lo stato dell'iscrizione
Postconditions	1. I commensali vengono notificati dell'esito della loro richiesta
Alternative flows	1. Se nessuna richiesta è presente, il sistema lo comunica all'oste

Diagramma dei casi d'uso

Recensione Commensale

Use case	RecensioneCommensale
ID	5
Brief description	L'oste lascia una recensione per ciascun commensale che ha partecipato alla cena.
Primary actors	Oste
Secondary actors	-
Preconditions	1. L'oste ha gestito la cena (incluso in GestioneIscrizioni)
Main flow	1. L'oste seleziona un commensale tra quelli che hanno partecipato 2. Inserisce una valutazione e un commento 3. Il sistema salva la recensione in modo pubblico
Postconditions	1. La recensione del commensale è visibile nel profilo dell'oste tramite notifica
Alternative flows	1. Nessuna recensione inserita: nessuna azione salvata

Iscrizione Cena

Use case	IscrizioneCena
ID	6
Brief description	Il commensale si iscrive a una cena pubblicata da un oste.
Primary actors	Commensale
Secondary actors	-
Preconditions	1. Il commensale è autenticato
Main flow	1. Il commensale seleziona una cena 2. Invia richiesta di partecipazione 3. Il sistema notifica l'oste
Postconditions	1. L'iscrizione risulta "in attesa" di approvazione
Alternative flows	1. Se la cena ha raggiunto il limite massimo, il sistema blocca l'iscrizione

Diagramma dei casi d'uso

Recensione Oste

Use case	RecensioneOste
ID	7
Brief description	Il commensale lascia una recensione all'oste dopo aver partecipato alla cena.
Primary actors	Commensale
Secondary actors	-
Preconditions	1. Il commensale ha partecipato a una cena
Main flow	1. Il commensale accede al riepilogo delle cene partecipate 2. Seleziona la cena e l'oste da recensire 3. Inserisce valutazione e commento 4. Il sistema salva la recensione
Postconditions	1. La recensione è visibile nel profilo dell'oste tramite notifica
Alternative flows	1. Nessuna recensione inviata: nessuna azione

Iscrizione Cena

Use case	Annullamento Partecipazione
ID	8
Brief description	Permette a un commensale di cancellare la propria partecipazione a una cena per la quale la sua richiesta era già stata accettata.
Primary actors	Commensale
Secondary actors	-
Preconditions	1. Il commensale è autenticato. 2. Esiste una partecipazione "Confermata" per il commensale a una cena futura.
Main flow	1. Il commensale visualizza l'elenco delle sue partecipazioni. 2. Seleziona la partecipazione che intende annullare 3. Il sistema chiede conferma. 4. L'utente conferma la volontà di annullare. 5. Il sistema aggiorna lo stato della partecipazione in "Annullato" 6. Il sistema notifica l'oste dell'annullamento.
Postconditions	1. La partecipazione del commensale risulta annullata. 2. Un posto per la cena corrispondente torna disponibile.
Alternative flows	1. Se L'utente annulla l'operazione alla richiesta di conferma, il sistema interrompe il flusso e lo stato della partecipazione non cambia. 2. Se non è possibile annullare la partecipazione (es. la cena è troppo vicina), il sistema mostra un messaggio di errore.

Diagramma dei casi d'uso

Annullamento Partecipazione

Use case	Annullamento Partecipazione
ID	8
Brief description	Permette a un commensale di cancellare la propria partecipazione a una cena per la quale la sua richiesta era già stata accettata.
Primary actors	Commensale
Secondary actors	-
Preconditions	1. Il commensale è autenticato. 2. Esiste una partecipazione "Confermata" per il commensale a una cena futura.
Main flow	1. Il commensale visualizza l'elenco delle sue partecipazioni. 2. Seleziona la partecipazione che intende annullare 3. Il sistema chiede conferma. 4. L'utente conferma la volontà di annullare. 5. Il sistema aggiorna lo stato della partecipazione in "Annullato" 6. Il sistema notifica l'oste dell'annullamento.
Postconditions	1. La partecipazione del commensale risulta annullata. 2. Un posto per la cena corrispondente torna disponibile.
Alternative flows	1. Se L'utente annulla l'operazione alla richiesta di conferma, il sistema interrompe il flusso e lo stato della partecipazione non cambia. 2. Se non è possibile annullare la partecipazione (es. la cena è troppo vicina), il sistema mostra un messaggio di errore.

Annullamento Cena

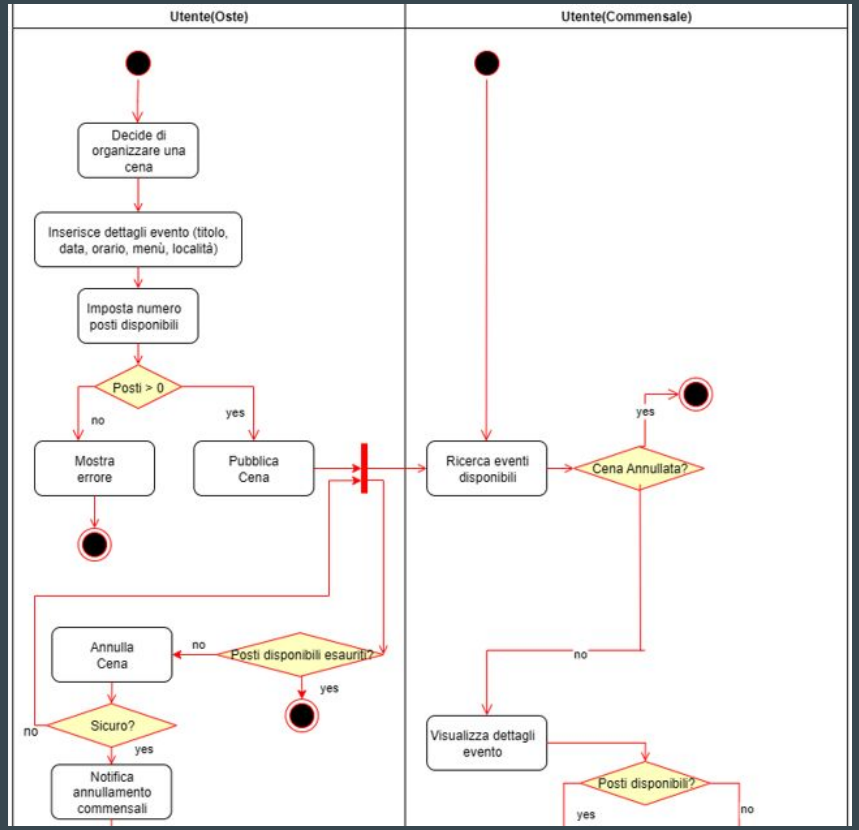
Use case	Annullamento Cena (da parte dell'Oste)
ID	9
Brief description	Permette a un commensale di cancellare la propria partecipazione a una cena per la quale la sua richiesta era già stata accettata.
Primary actors	Oste
Secondary actors	-
Preconditions	1. L'oste è autenticato. 2. Esiste una cena futura creata dall'oste.
Main flow	1. L'oste accede alla lista delle sue cene. 2. Seleziona la cena che intende annullare. 3. Il sistema chiede conferma dell'azione. 4. L'oste conferma la volontà di annullare la cena. 5. Il sistema aggiorna lo stato della cena in "Annullata". 6. Il sistema invia una notifica a tutti i commensali che avevano la richiesta di partecipazione accettata per quella cena.
Postconditions	1. Lo stato della cena è "Annullata". 2. I commensali che avevano accettato la partecipazione sono informati dell'annullamento.
Alternative flows	1. L'oste non conferma l'annullamento: Il sistema annulla l'operazione e la cena rimane nello stato precedente. 2. La cena è già iniziata o conclusa: Il sistema mostra un messaggio di errore e l'annullamento non è consentito.

Diagramma dei casi d'uso

Visualizzazione Notifiche

Use case	**Visualizzazione Notifiche**
ID	10
Brief description	Consente a un utente (Oste o Commensale) di visualizzare le notifiche recenti relative alle proprie attività nel sistema (es. nuove richieste, esiti di iscrizioni, annullamenti di cene, ecc.).
Primary actors	Oste, Commensale
Secondary actors	-
Preconditions	1. L'utente è autenticato.
Main flow	1. L'utente seleziona l'opzione per visualizzare le notifiche. 2. Il sistema recupera le notifiche associate all'utente (secondo il vincolo delle 5 più recenti). 3. Il sistema visualizza l'elenco delle notifiche. 4. Il sistema segna le notifiche come "lette".
Postconditions	1. L'utente ha visualizzato le sue notifiche. 2. Le notifiche sono segnate come "lette" nel database.
Alternative flows	1. Nessuna notifica presente: Il sistema visualizza un messaggio come "Nessuna nuova notifica".

Diagramma di attività



Descrive il flusso di lavoro (workflow) di processi complessi, come l'intero ciclo di vita di una cena dall'organizzazione alla partecipazione, mostrando le azioni e le decisioni prese dagli attori.

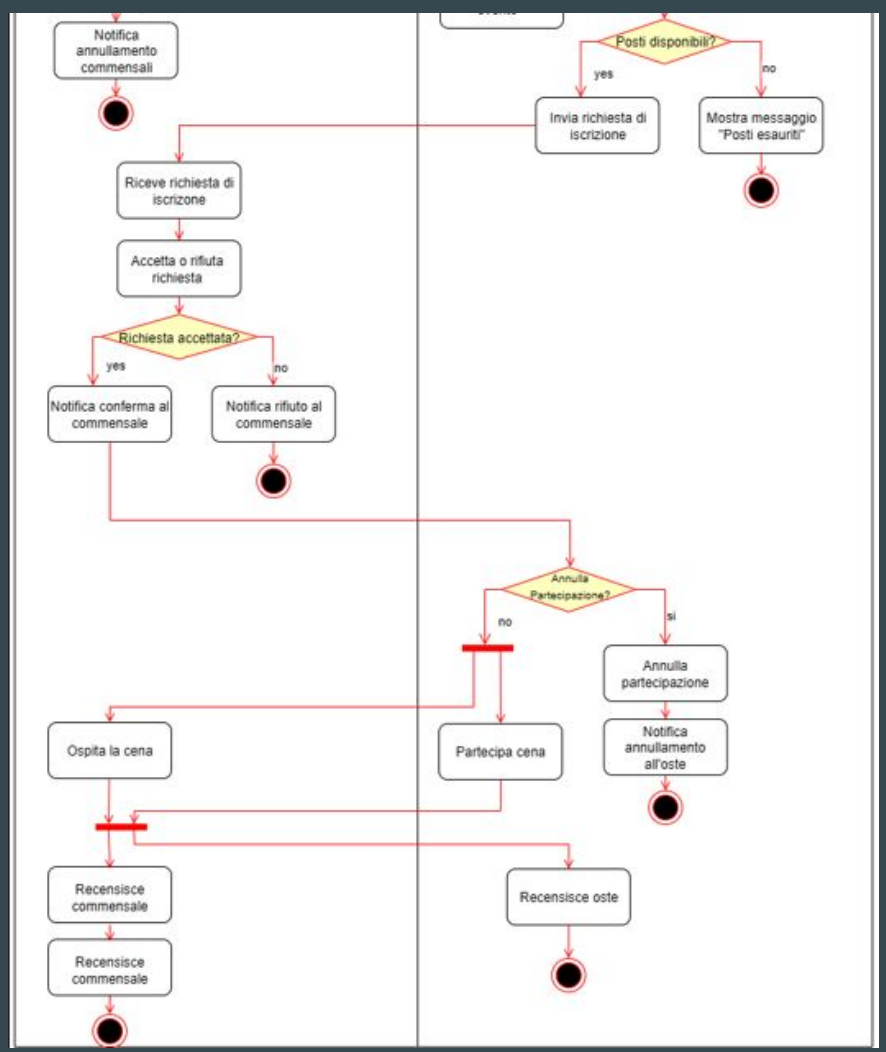


Diagramma di attività

Note a margine:
- Un Utente può essere sia Oste che Commensale
- L'Oste può decidere di annullare la Cena in qualunque istante temporale prima che la cena termini i posti disponibili

Raccolta dei requisiti

Funzionali

ID	Nome	Descrizione
RF1	Registrazione Utente	Un utente deve potersi registrare al sistema fornendo nome, cognome, email e password. Il sistema deve impedire registrazioni con email già in uso.
RF2	Autenticazione Utente	Un utente registrato deve poter accedere al sistema tramite email e password. Il sistema deve negare l'accesso in caso di credenziali errate.
RF3	Creazione Cena	Un utente autenticato (Oste) deve poter creare una nuova cena, specificandone i dettagli come titolo, descrizione, data, località, menù e numero di posti disponibili.
RF4	Pubblicazione Cena	Il sistema deve rendere una cena creata visibile agli altri utenti (Commensali).
RF5	Iscrizione a Cena	Un utente autenticato (Commensale) deve poter inviare una richiesta di partecipazione per una cena disponibile. (Il numero di posti disponibili di una specifica cena viene decrementato, a seguito una richiesta di partecipazione da parte di un utente, solo se l'oste accetta la richiesta di partecipazione)
RF6	Gestione Iscrizioni	L'Oste deve poter visualizzare le richieste di partecipazione per le sue cene e decidere se accettarle o rifiutarle.
RF7	Notifica Esito Richiesta	Il sistema deve notificare il Commensale sull'esito (accettato o rifiutato) della sua richiesta di partecipazione.
RF8	Annullamento Partecipazione	Un Commensale la cui richiesta è stata accettata deve poter annullare la propria partecipazione a una cena futura. Il sistema deve notificare l'Oste.
RF9	Annullamento Cena	L'Oste deve poter annullare una cena futura da lui creata. Il sistema deve notificare tutti i partecipanti confermati.
RF10	Recensione Oste-Commensale	Al termine di una cena, l'Oste deve poter lasciare una recensione per ogni Commensale partecipante. (Il testo non può superare i 255 caratteri) (L'oste può recensire ogni commensale che ha partecipato)
RF11	Recensione Commensale-Oste	Al termine di una cena, un Commensale deve poter lasciare una recensione per l'Oste. (Il testo non può superare i 255 caratteri) (Ogni commensale può recensire l'oste)
RF12	Visualizzazione Notifiche	Un utente deve poter visualizzare le sue 5 notifiche più recenti. Dopo la visualizzazione, le notifiche devono essere segnate come lette.

Non Funzionali

ID	Nome	Descrizione
RNF1	Usabilità	L'interfaccia deve essere chiara e intuitiva sia per l'Oste che per il Commensale, guidando l'utente attraverso i flussi principali (creazione, iscrizione, gestione).
RNF2	Affidabilità	Il sistema deve gestire correttamente gli stati degli oggetti (Cena, Richiesta, Partecipazione) come definito nei diagrammi di stato, garantendo la consistenza dei dati.
RNF3	Performance	Le query per la visualizzazione di cene, richieste e notifiche devono essere eseguite in tempi rapidi per non compromettere l'esperienza utente.
RNF4	Vincoli Implementativi	- L'applicazione lato server deve essere eseguita all'interno di un container Docker. - L'autore di una recensione e il recensito devono aver partecipato alla stessa cena. L'autore di una recensione e il recensito devono aver partecipato alla stessa cena.
RNF5	Sicurezza	L'accesso alle funzionalità riservate (es. creare una cena, gestire iscrizioni) deve essere protetto e consentito solo ad utenti autenticati e autorizzati (Oste/Commensale).

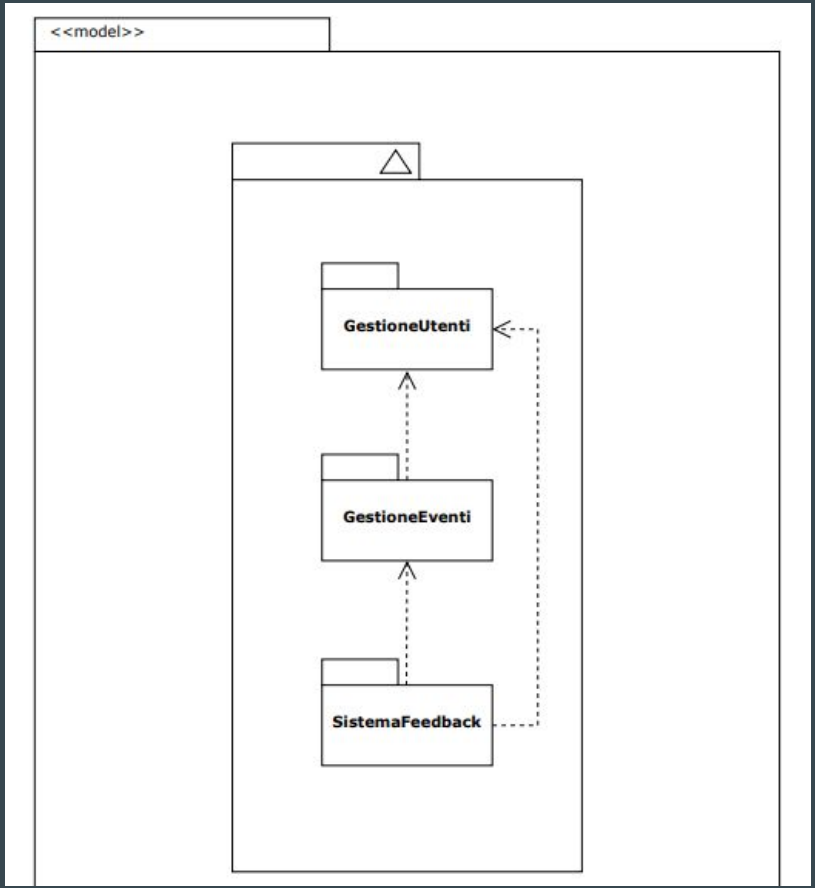
Raccolta dei Requisiti: è il documento principale di questa fase.
Definisce in modo formale i requisiti funzionali e non funzionali del sistema

Progettazione

Per questa fase abbiamo utilizzato diagrammi come:

- Diagramma delle classi
- Diagramma delle attività
- Diagramma di stato

Diagramma di Package



Mostra l'organizzazione logica del sistema in macro-componeni

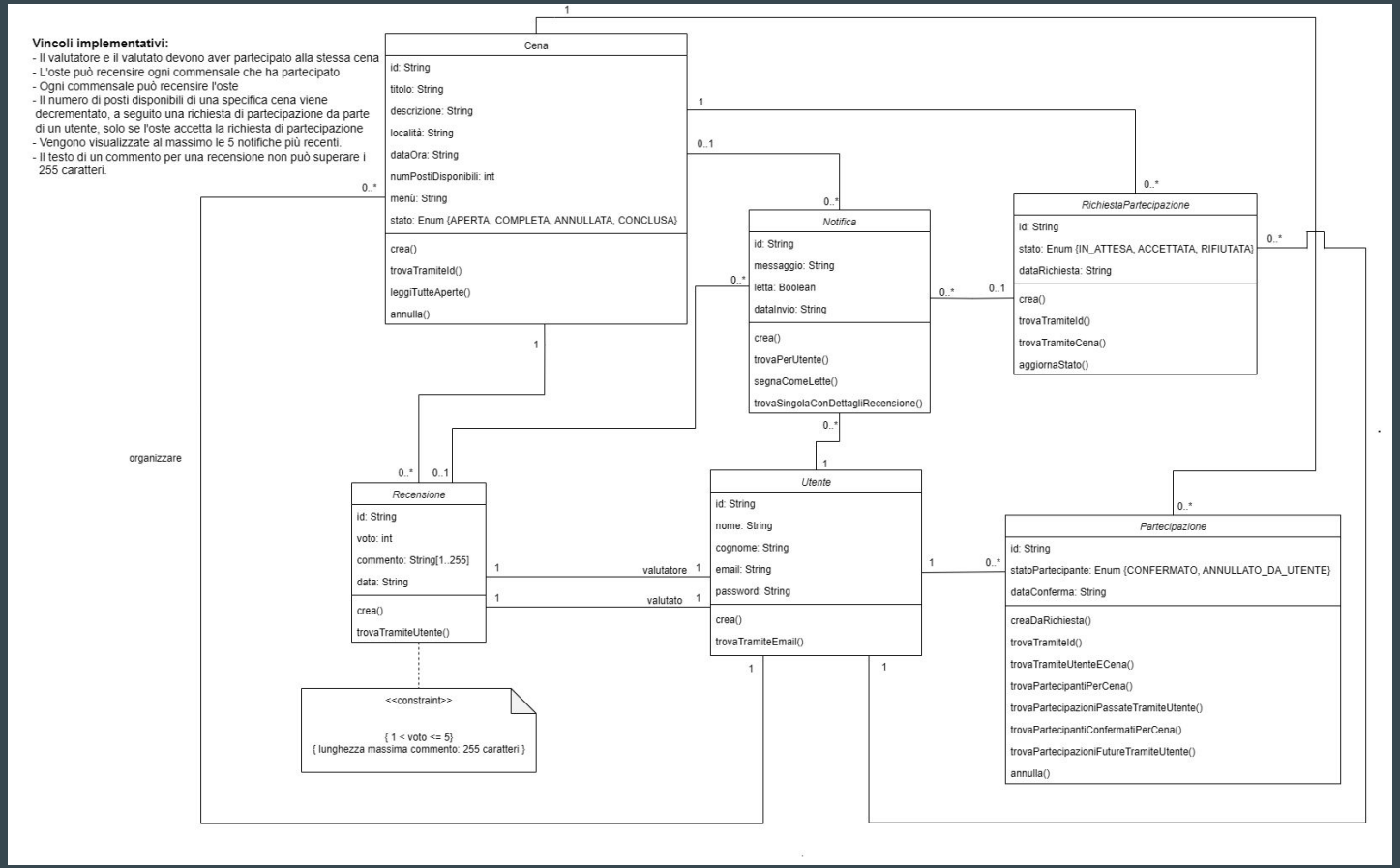
Osservazioni:

1) GestioneEventi dipende da GestioneUtenti perché classi come Cena e Partecipazione (dentro a GestioneEventi) hanno bisogno di conoscere le classi Oste e Commensale (dentro GestioneUtenti)

2)Allo stesso modo SistemaFeedback dipende da GestioneEventi perché le classi Recensioni e Notifica devono sapere a quale Utente si riferiscono

3) SistemaFeedback dipende da GestioneEventi perché una Recensione è legata a una Cena, e una notifica può riferirsi a una Cena o a una RichiestaPartecipazione

Diagramma delle classi (vista strutturale)

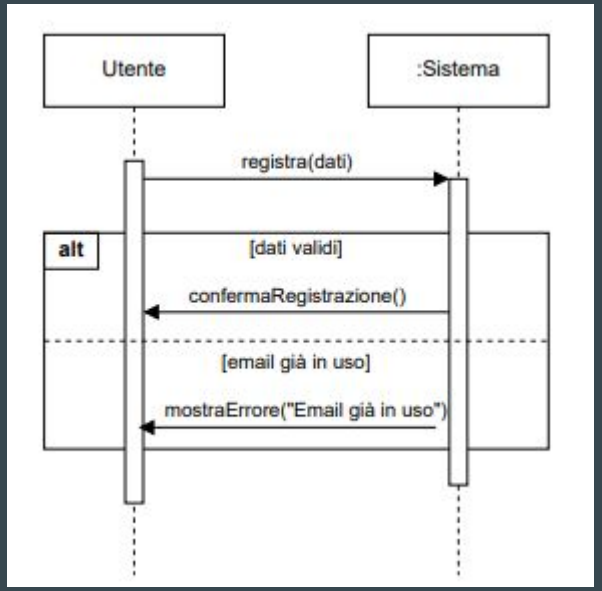


Questo diagramma definisce lo scheletro del sistema rappresentando classi, associazioni, attributi, ecc...

Diagramma di Sequenza

Descrive il comportamento dinamico del sistema, mostrando come gli oggetti delle varie classi collaborano tra loro scambiandosi messaggi in una sequenza temporale per realizzare un caso d'uso.

Registrazione



Login

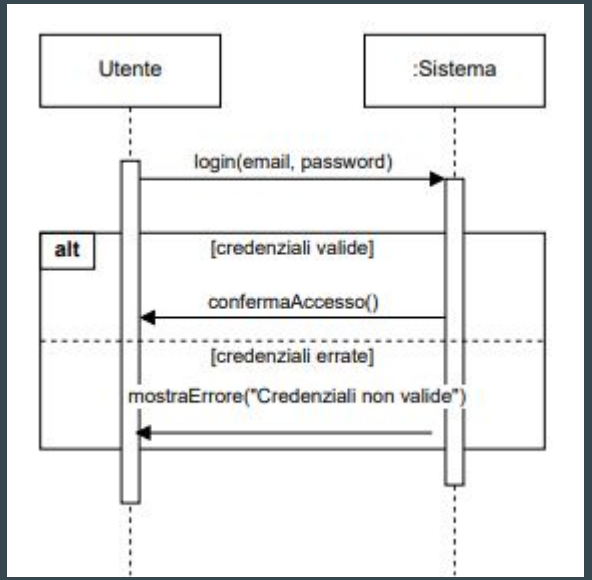


Diagramma di Sequenza

Creazione Cena

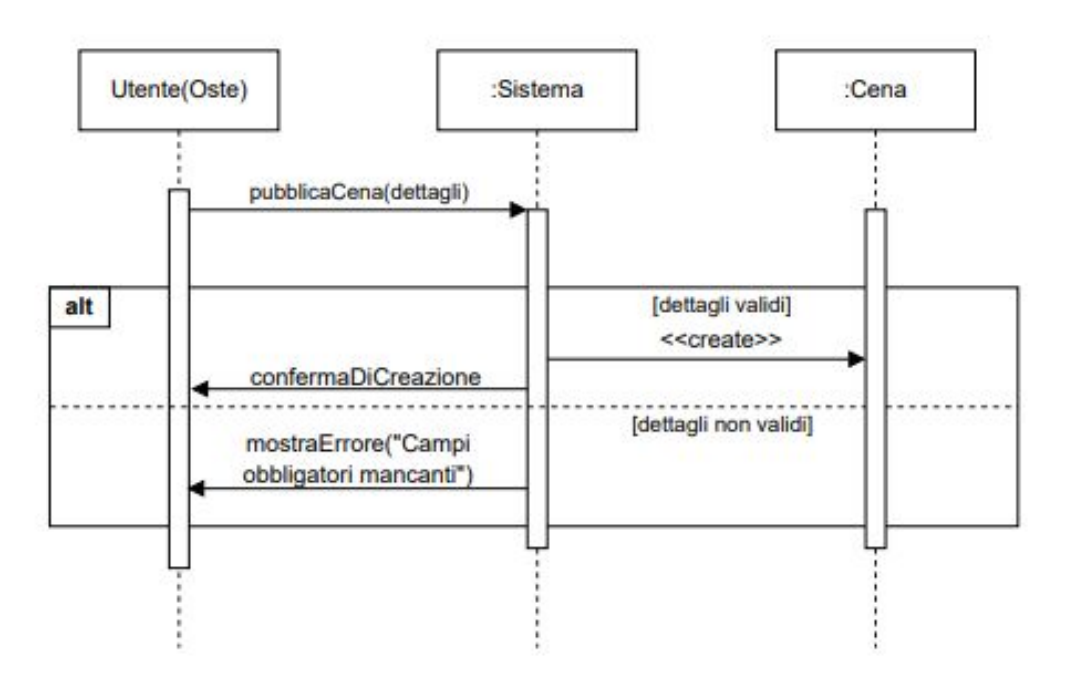


Diagramma di Sequenza

Iscrizione Cena

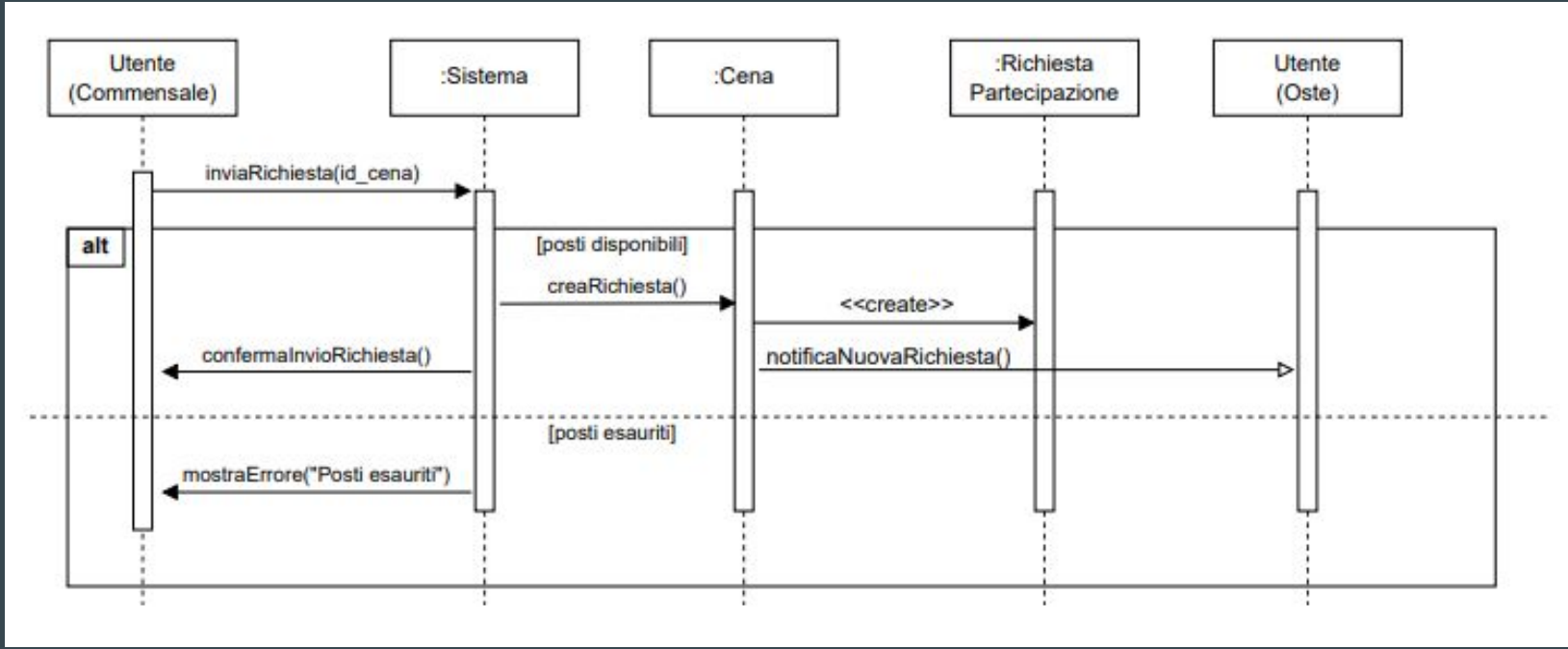


Diagramma di Sequenza

Gestione Iscrizioni (a)

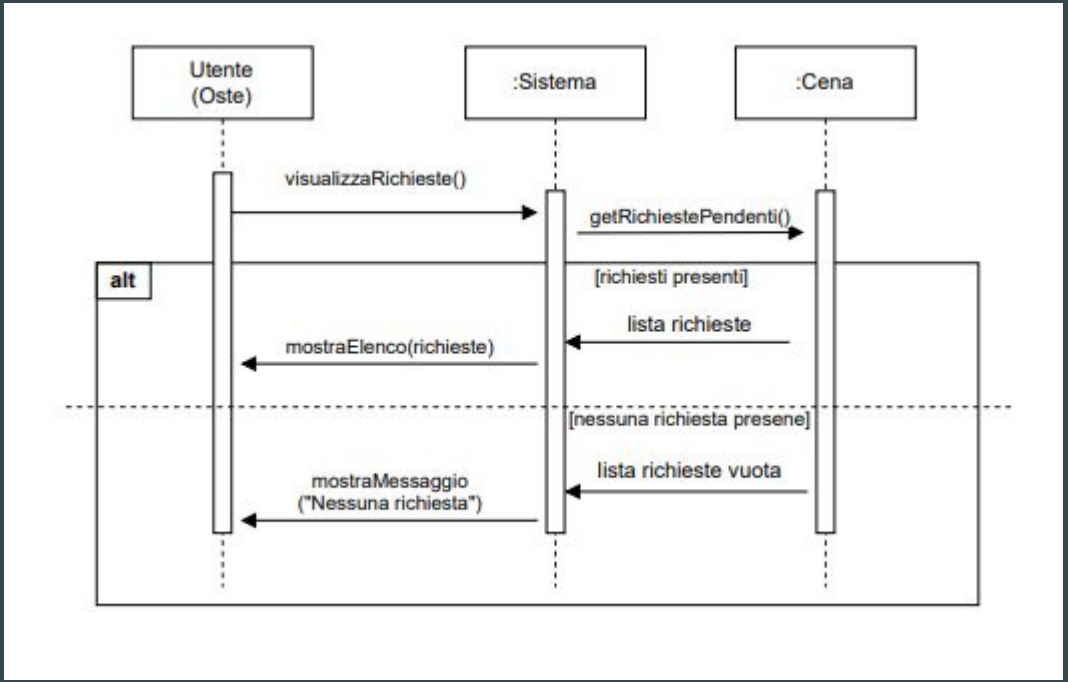


Diagramma di Sequenza

Gestione Iscrizioni (b)

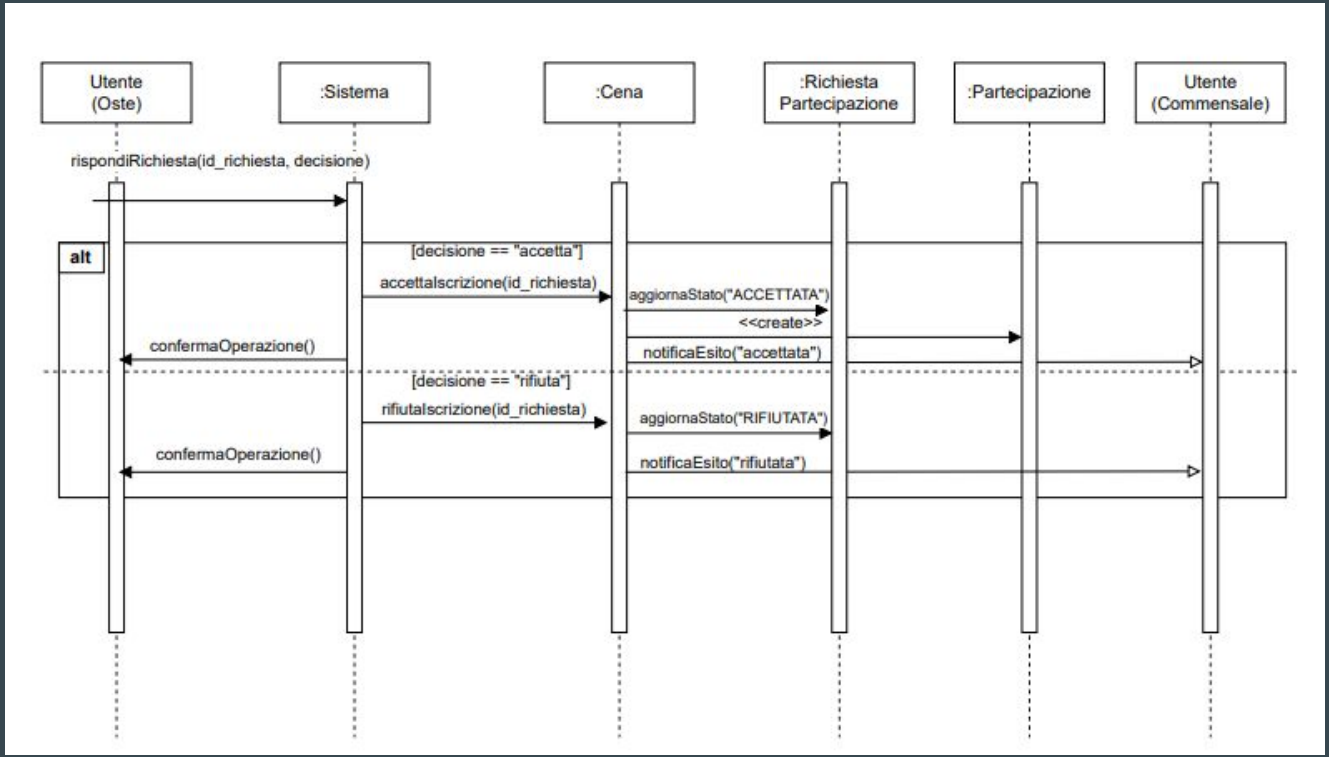


Diagramma di Sequenza

Annulla Partecipazione

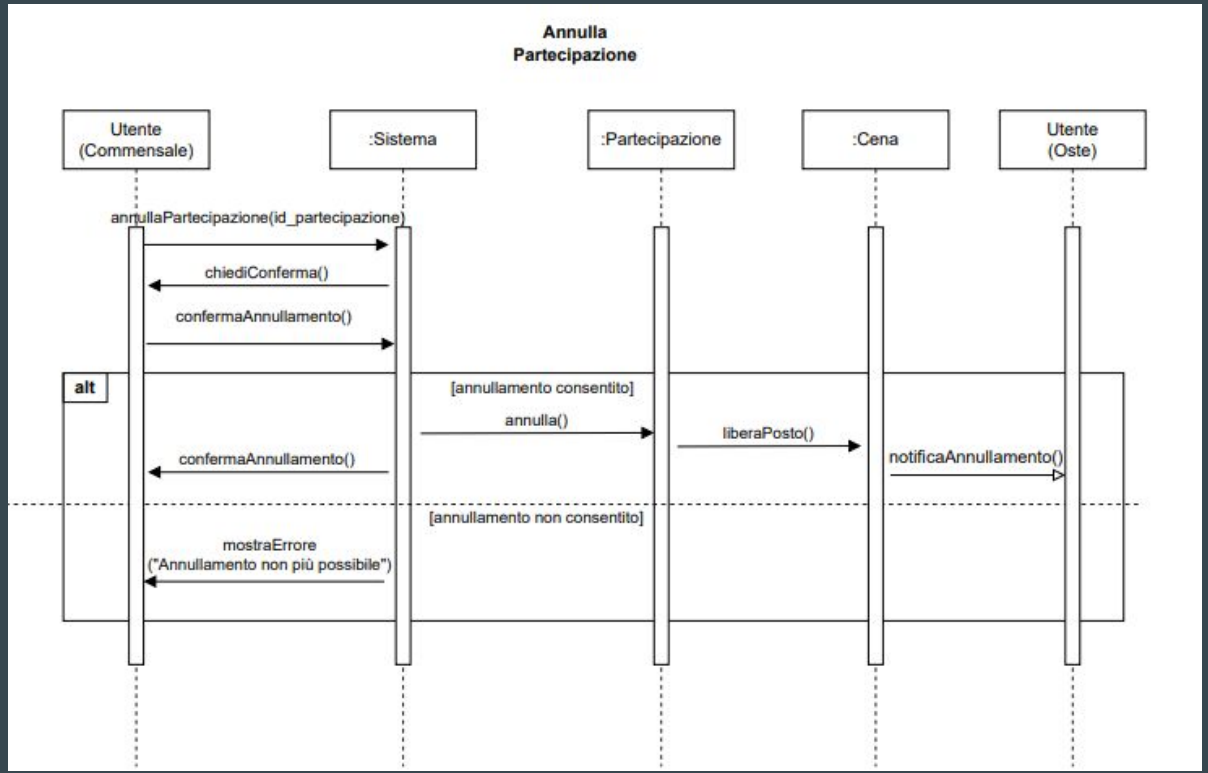


Diagramma di Sequenza

Annulla Partecipazione

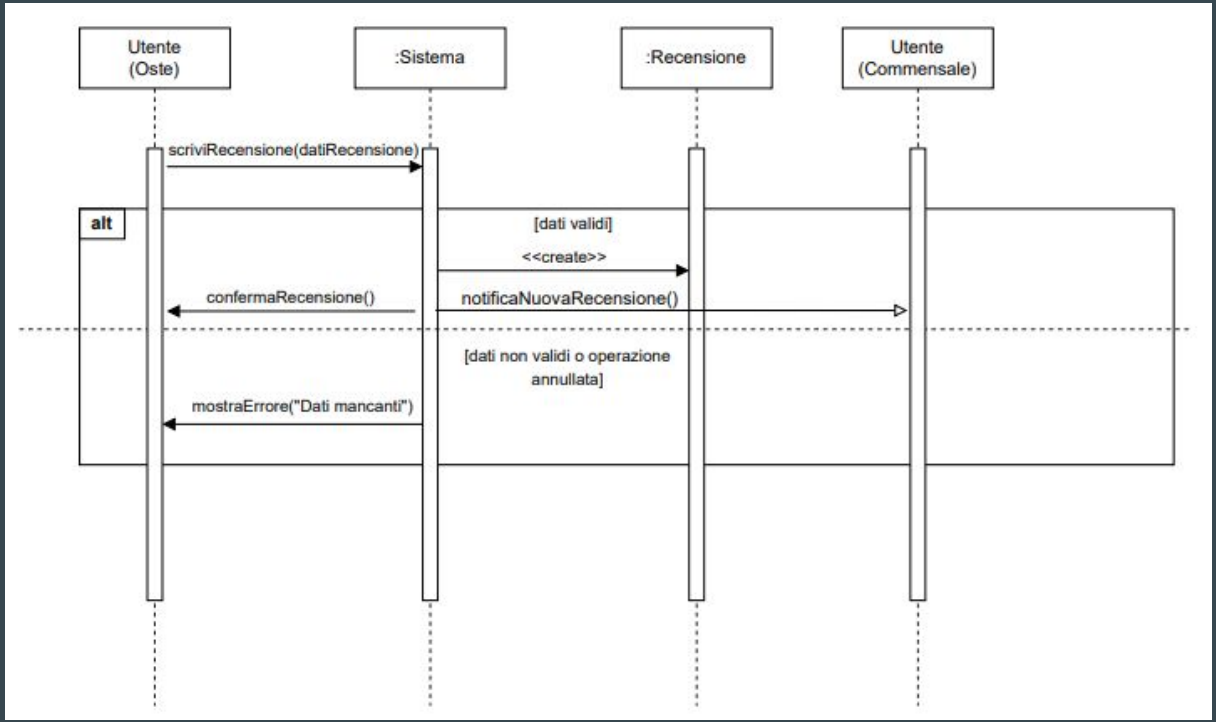


Diagramma di Sequenza

Recensione Commensale

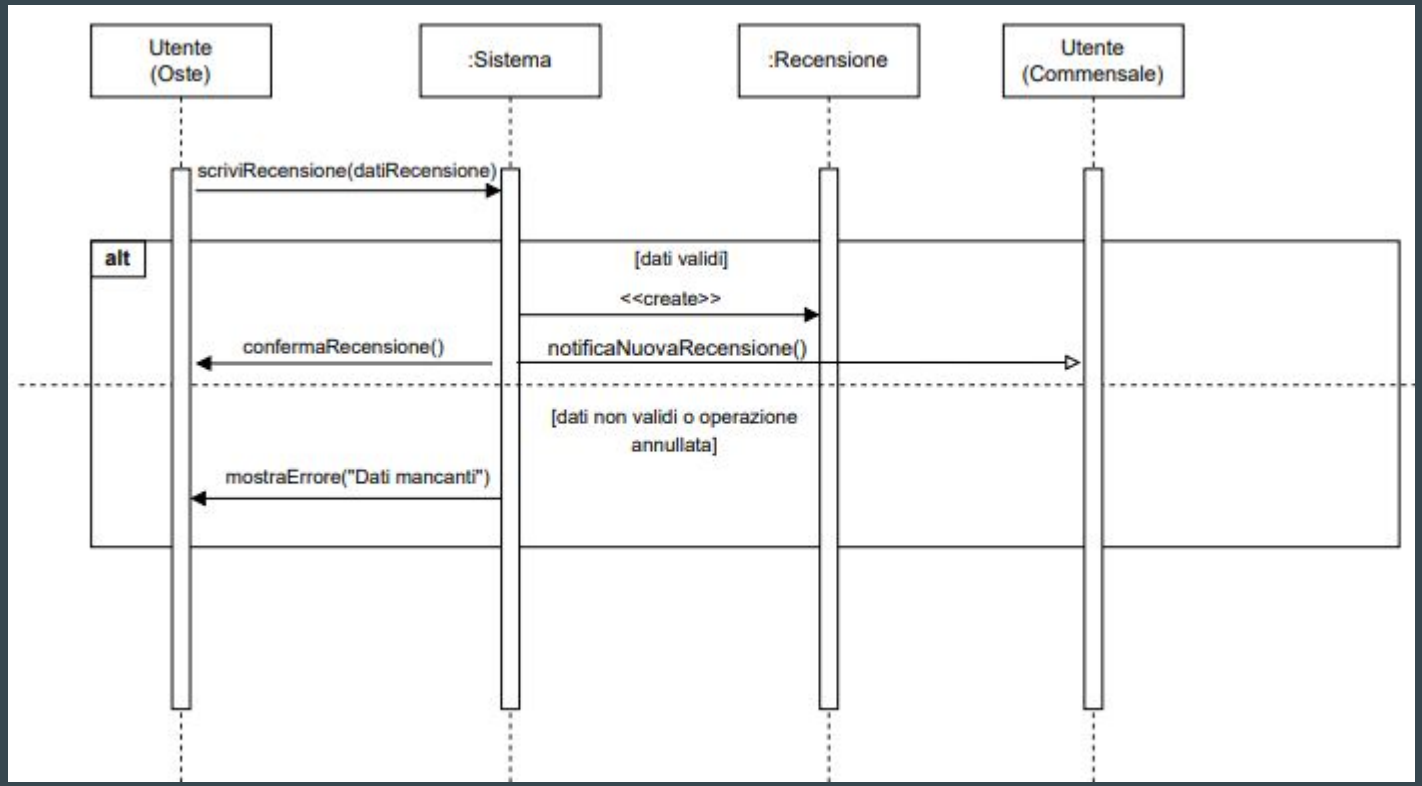


Diagramma di Sequenza

Recensione Oste

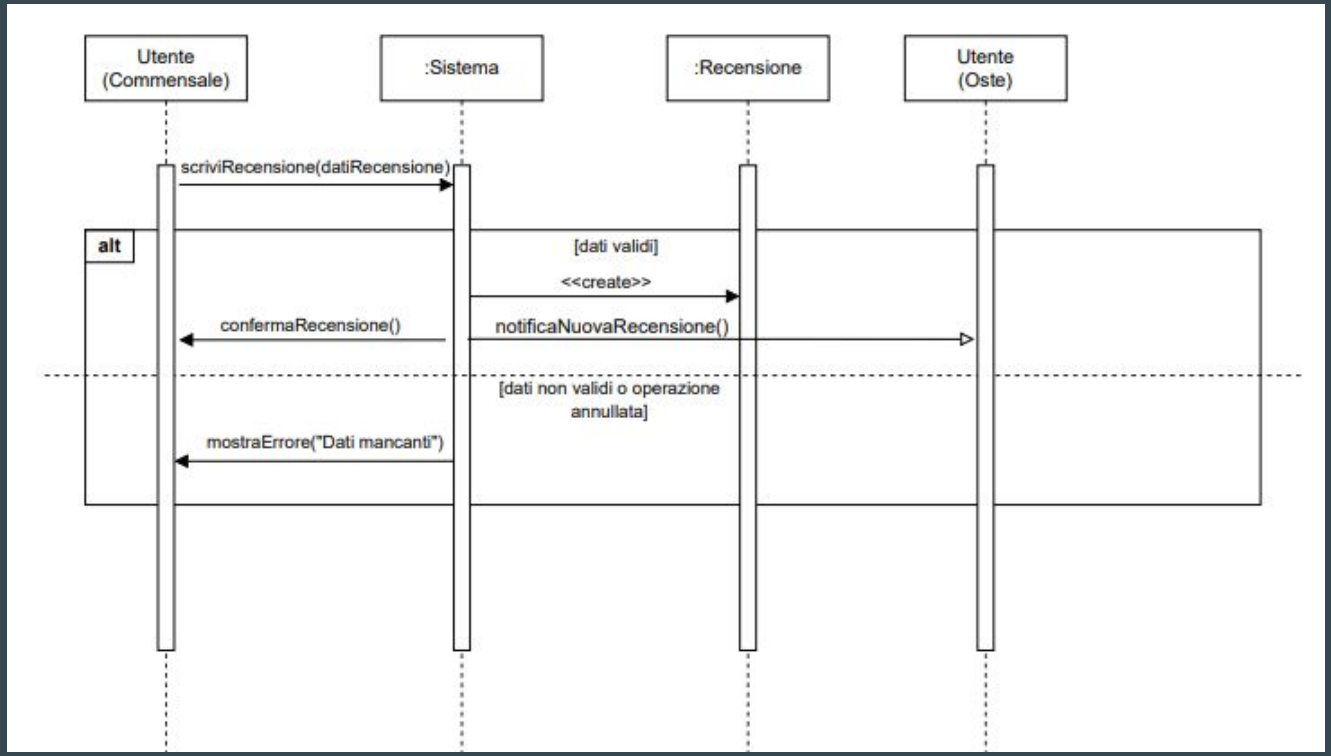


Diagramma di Sequenza

Annulla Cena

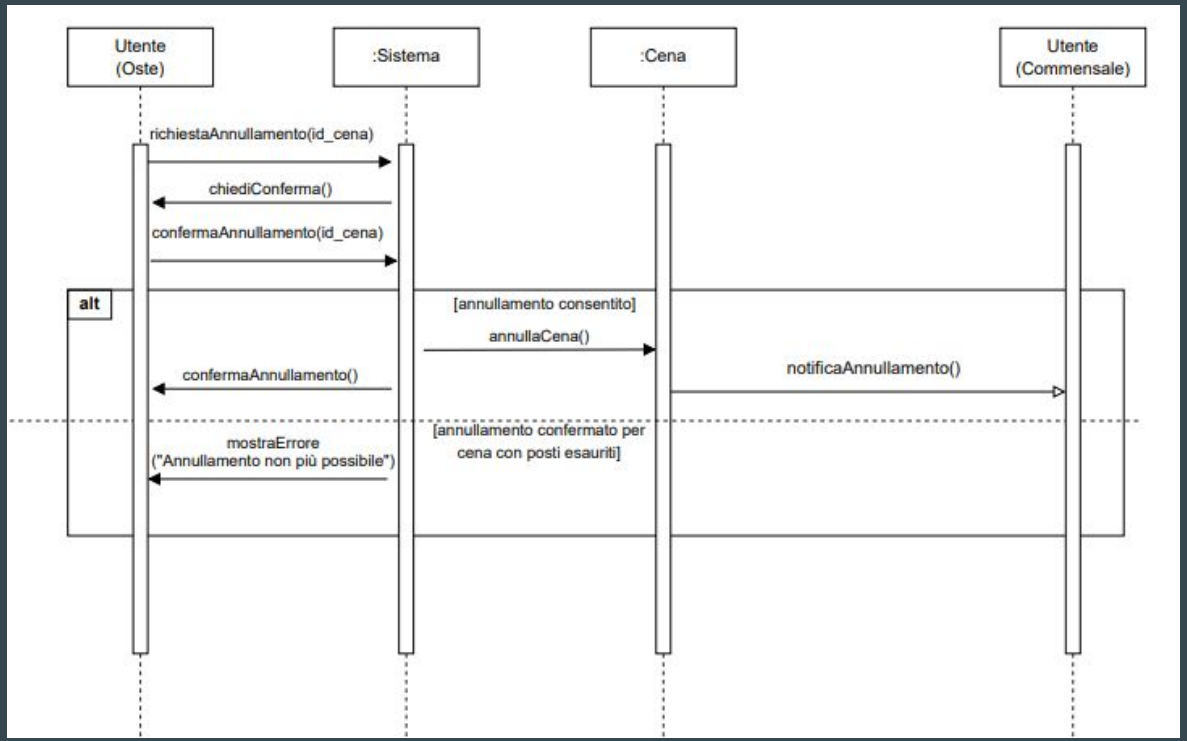


Diagramma di Sequenza

Visualizzazione Notifica

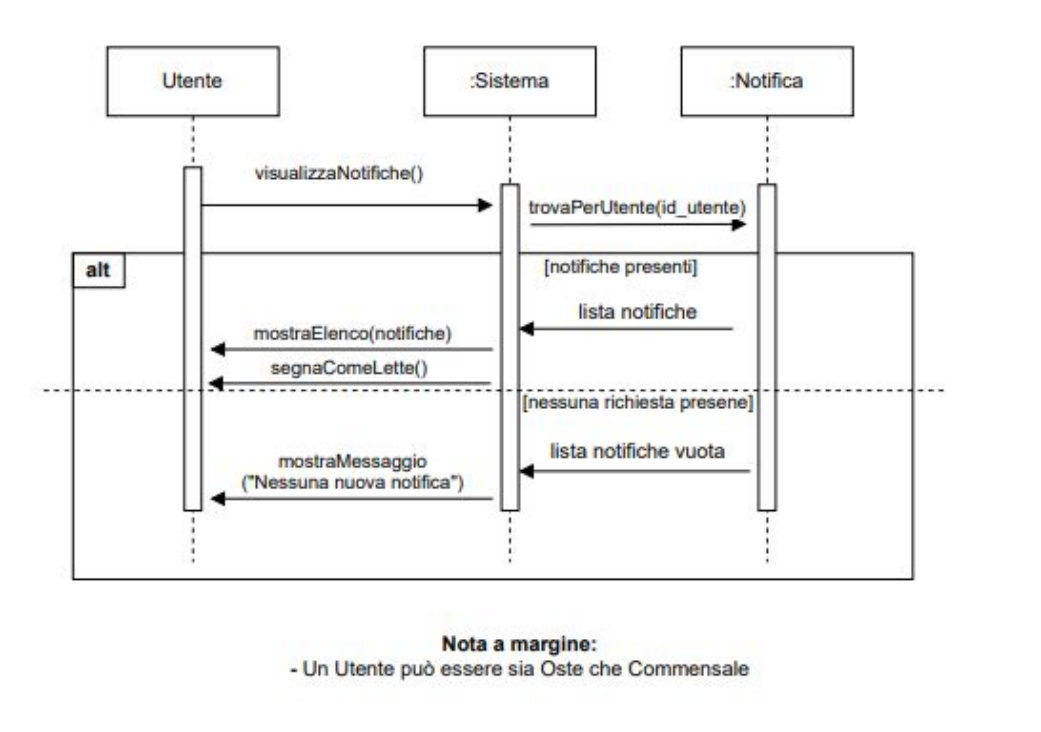


Diagramma di Stato

Modella il ciclo di vita di oggetti oggetti come Cena, mostrando i diversi stati in cui possono trovarsi e le transizioni che li fanno cambiare (macchina a stato).

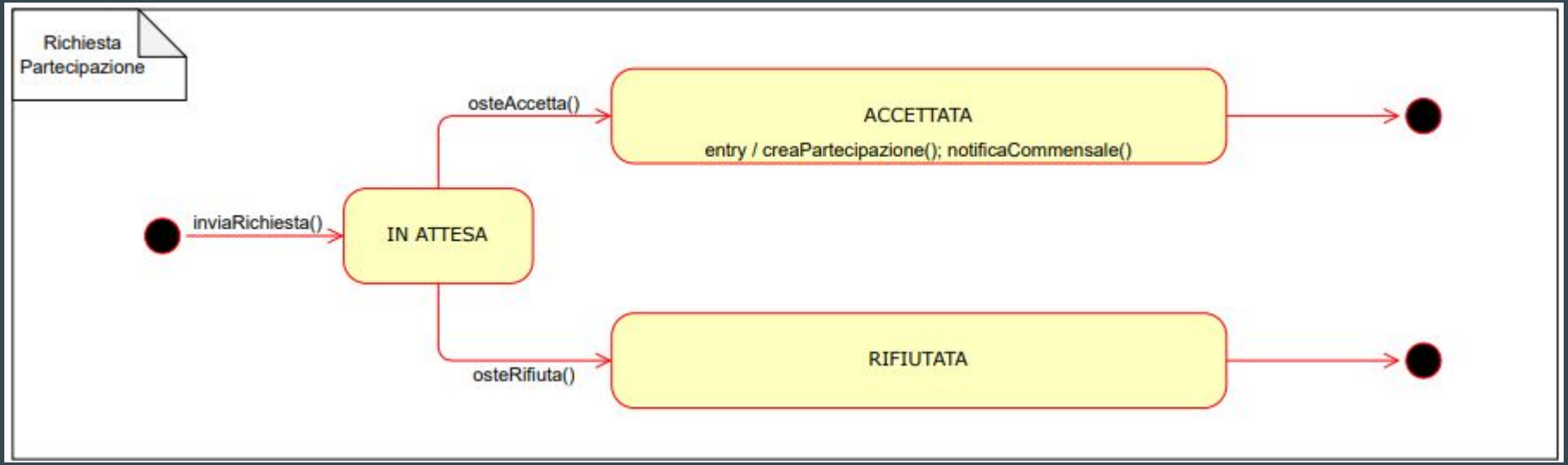
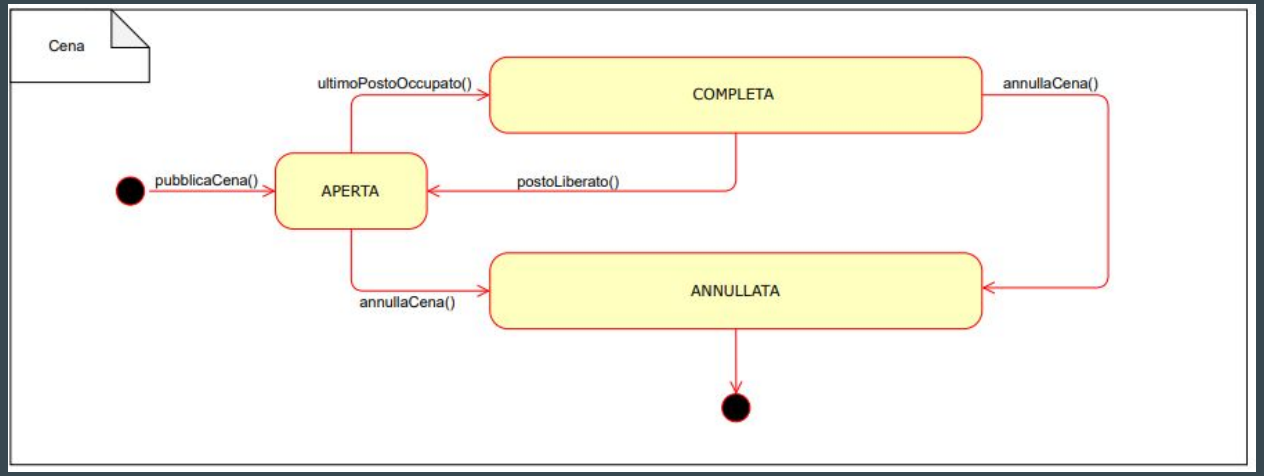


Diagramma di Stato



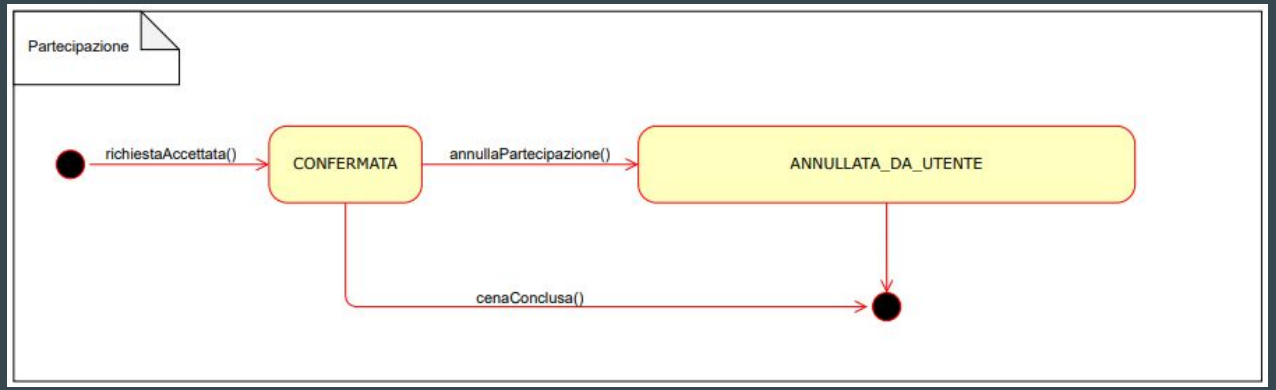
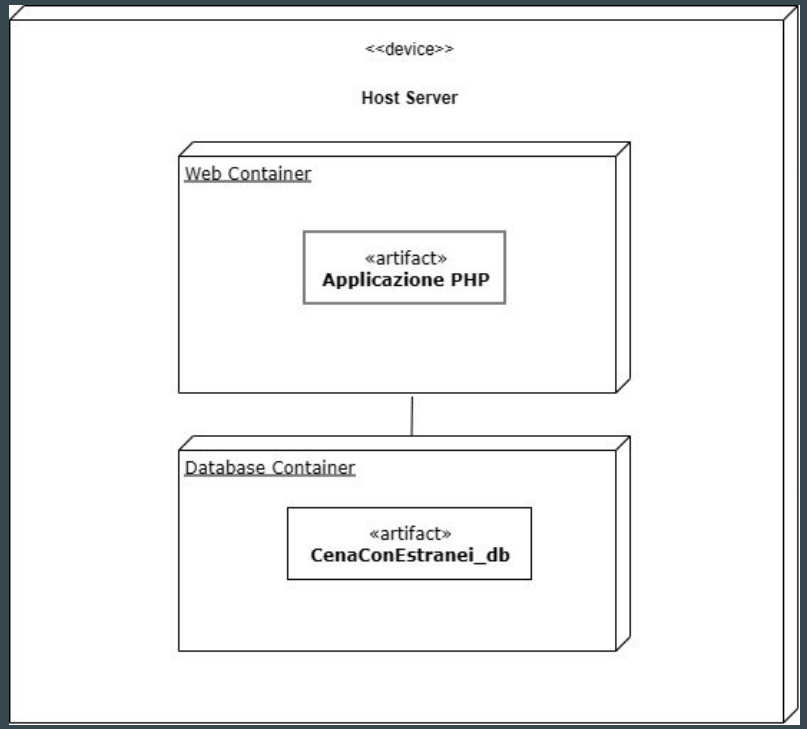


Diagramma di Distribuzione

Progetta l'architettura fisica del sistema, mostrando come i componenti software (l'applicazione PHP, il database) vengono distribuiti sui nodi hardware (i container Docker).



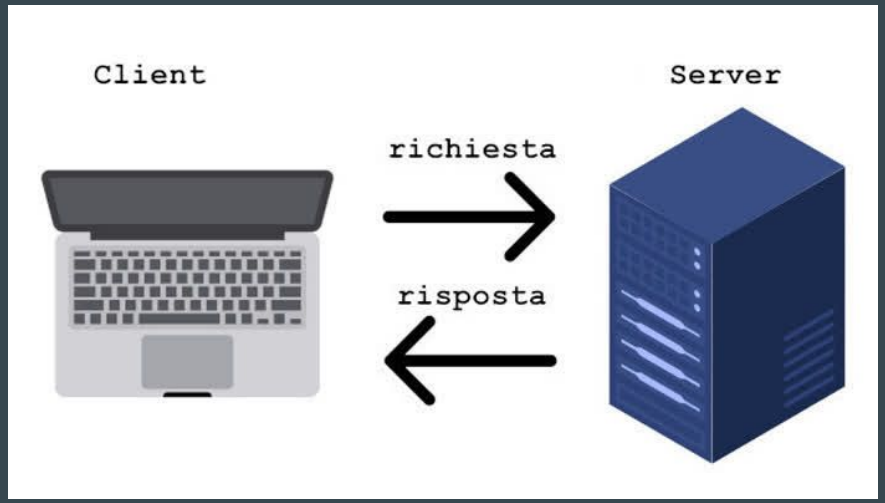
Implementazione

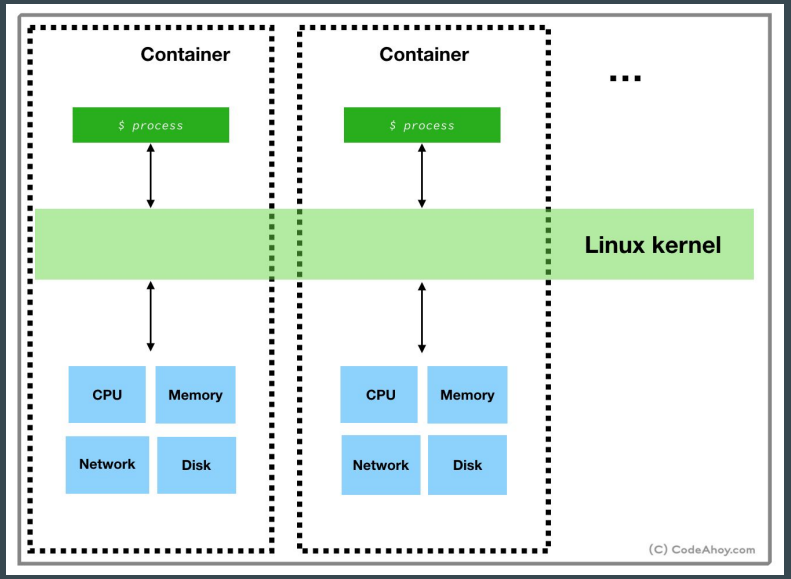
Aspetti:

- Architettura
- Tecnologie utilizzate
- Design pattern applicati
- Progettazione del Database
- Misure di Sicurezza

Architettura

Il progetto segue un'architettura client-server basata su un modello a più livelli:
- Client (Frontend)
- Server (Backend)
- Database (DataLayer)





Configurazioni Docker

docker-compose.yml

```yaml
version: '3.8'
services:
  web:
    build: .
    container_name: cena-web-server
    ports:
      - "8000:80"
    volumes:
      - ./src:/var/www/html/src
      - ./public:/var/www/html/public
    environment:
      - DB_HOST=db
      - DB_NAME=${DB_NAME}
      - DB_USER=${DB_USER}
      - DB_PASS=${DB_PASS}
      - JWT_SECRET_KEY=${JWT_SECRET_KEY}
    depends_on:
      - db
  db:
    image: mysql:8.0
    container_name: cena-db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASS}
    ports:
      - "3306:3306"
    volumes:
      - ./schema.sql:/docker-entrypoint-initdb.d/init.sql
      - db_data:/var/lib/mysql
volumes:
  db_data:
```

L'utilità di usare docker compose è definire un'intera applicazione multi-container in un unico file per poi avviare e collegare tutti i servizi insieme con un singolo comando

.env

```
# Configurazione del Database
DB_NAME=CenaConEstranei_db
DB_USER=user
DB_PASS=123@Password@123
MYSQL_ROOT_PASSWORD=123@Password@123

# Chiave per l'applicazione
JWT_SECRET_KEY=una_frase_segreta_molto_molto_lunga_e_difficile
```

Dockerfile

```
# Usa un'immagine ufficiale di PHP con Apache
FROM php:8.2-apache

# Installa le estensioni PHP necessarie per connettersi a MySQL
RUN docker-php-ext-install pdo pdo_mysql

# Abilita mod_rewrite di Apache per gli URL "puliti"
RUN a2enmod rewrite

# Copia l'intero progetto nella root del server web del container
COPY . /var/www/html/

# Imposta la DocumentRoot sulla cartella /public per sicurezza
RUN sed -ri -e 's!/var/www/html!/var/www/html/public!g' /etc/apache2/sites-available/*.conf
```

Tecnologie Utilizzate

Frontend

- HTML5/CSS
- Javascript/JQuery
- Bootstrap

Logica Centralizzata: file app.js agisce da "cervello" del frontend:

- gestione delle viste (il router object).
- Le chiamate AJAX al backend.
- La manipolazione dinamica del DOM

Backend

- PHP
- Apache

Logica Centralizzata: il file index.php fa da FrontController e gestisce tramite il router le richieste

Database

- MySQL
- PDO

Viene utilizzato un database MySQL per la storicizzazione dei dati dell'applicativo, libreria PDO usata per accedere dati

Alberatura e Design pattern

Design Pattern Applicati
Model-View-Controller (MVC): separare le responsabilità tra Model, View e Controller

Front Controller: Tutte le richieste API vengono reindirizzate dal web server a un unico punto di ingresso (index.php), che inizializza l'applicazione e un Router per smistare la richiesta al controller corretto.

Singleton: Utilizzato nella classe Database per assicurare che esista una sola istanza della connessione al database per tutta la durata della richiesta, ottimizzando le risorse.

Frontend

```
$(document).ready(function () {
    const API_BASE_URL = '/api';
    let currentUser = null; // Memorizza i dati dell'utente loggato

    const router = {
        showView: (viewId) => {
            $('main > section').hide();
            $(`#${viewId}`).show();
        },
        updateNav: () => {
            const token = localStorage.getItem('jwtToken');
            if (token) {
                try {
                    // Decodifichiamo il payload del token per leggere i dati utente
                    const payload = JSON.parse(atob(token.split('.')[1]));
                    currentUser = payload.data;
                } catch (e) {
                    handleLogout();
                    return; // Se il token non è valido, esegui il logout
                }

                // Se c'è un token, mostra i link per gli utenti autenticati e nascondi quelli per i non autenticati
                $('#login-link, #register-link').hide();
                $('#logout-link, #home-link, #create-dinner-link, #my-dinners-link, #my-participations-link, #notifications-li').show();
            } else {
                // Se non c'è un token, mostra i link per i non autenticati e nascondi quelli per gli autenticati
                currentUser = null;
                $('#login-link, #register-link').show();
                $('#logout-link, #create-dinner-link, #home-link, #my-dinners-link, #my-participations-link, #notifications-li').hide();
            }
        }
```

```
function fetchDinnerDetails(dinnerId) {
    $.ajax({
        url: `${API_BASE_URL}/cene/${dinnerId}`,
        method: 'GET',
        beforeSend: (xhr) => {
            const token = localStorage.getItem('jwtToken');
            if (token) {
                xhr.setRequestHeader('Authorization', 'Bearer ' + token);
            }
        },
        success: (dinner) => {
            renderDinnerDetails(dinner);
            router.showView('dinner-detail-view');
        },
        error: (err) => {
            showToast('Errore nel caricamento dei dettagli della cena.', 'error');
            router.showView('dinner-list-view');
        }
    });
}

//Funzion (local function) renderDinnerDetails(dinner: any): void
function renderDinnerDetails(dinner) {
    const container = $('#dinner-details-container');
    container.empty();
    const isHost = currentUser && currentUser.id == dinner.id_oste;
    const canParticipate = currentUser && !isHost && dinner.numPostiDisponibili > 0 && dinner.stato === 'APERTA' && !dinner.stato_richiesta_utente;
    const detailsHtml = `
        <div class="card p-4">
            <h2>${dinner.titolo}</h2>
            <p><strong>Descrizione:</strong> ${dinner.descrizione}</p>
            <p><strong>Menu:</strong> ${dinner.menu}</p>
            <p><strong>Data:</strong> ${new Date(dinner.dataOra).toLocaleString('it-IT')}</p>
            <p><strong>Località:</strong> ${dinner.localita}</p>
            <p><strong>Posti disponibili:</strong> ${dinner.numPostiDisponibili}</p>
            ${canParticipate
            ? `<button class="btn btn-primary mt-3 participate-btn" data-id="${dinner.id}">Invia richiesta di partecipazione</button>`
            : ''}
        </div>`;
    container.append(detailsHtml);
}
```

```
$(document).on('click', '.dinner-card', function () {
    const dinnerId = $(this).data('id');
    fetchDinnerDetails(dinnerId);
});
```

Backend

```php
$router = new Router();

// --- Rotte Utente ---
$router->post('/api/registrati', [UserController::class, 'registrati']);
$router->post('/api/login', [UserController::class, 'login']);

// --- Rotte Cene ---
$router->post('/api/cene', [DinnerController::class, 'crea']);
$router->get('/api/cene', [DinnerController::class, 'leggiTutte']);
$router->get('/api/cene/mie', [DinnerController::class, 'leggiCeneOrganizzate']);
$router->get('/api/cene/{id}', [DinnerController::class, 'leggiSingola']);
$router->post('/api/cene/annulla/{id}', [DinnerController::class, 'annulla']);

// --- Rotte Richieste di Partecipazione ---
$router->post('/api/richieste', [ParticipationController::class, 'richiediPartecipazione']);
$router->get('/api/cene/{id}/richieste', [ParticipationController::class, 'leggiRichiestePerCena']);
$router->put('/api/richieste/{id}', [ParticipationController::class, 'gestisciRichiesta']);
$router->get('/api/partecipazioni/mie/passate', [ParticipationController::class, 'leggiPartecipazioniPassateUtente']);
$router->get('/api/partecipazioni/mie/future', [ParticipationController::class, 'leggiPartecipazioniFutureUtente']);

$router->get('/api/cene/{id}/partecipanti', [ParticipationController::class, 'leggiPartecipantiCena']);
$router->post('/api/partecipazioni/annulla/{id}', [ParticipationController::class, 'annullaPartecipazione']);

// --- Rotte Recensioni ---
$router->post('/api/recensioni', [ReviewController::class, 'crea']);
$router->get('/api/utenti/{id}/recensioni', [ReviewController::class, 'leggiRecensioniPerUtente']);

// --- Rotte Notifiche ---
$router->get('/api/notifiche', [NotificationController::class, 'leggiPerUtente']);
$router->post('/api/notifiche/leggi', [NotificationController::class, 'marcaLette']);
$router->get('/api/notifiche/{id}', [NotificationController::class, 'leggiSingola']);

$url = $_SERVER['REQUEST_URI'];
$router->instrada($url);
```

```php
<?php
namespace App\Core;

class Router {
    private $routes = [];

    private function aggiungiRotta($method, $url, $handler) {
        $urlRegex = preg_replace('/\{([a-zA-Z0-9_]+)\}/', '(?P<$1>[a-zA-Z0-9_]+)', $url);
        $this->routes[$method]['#^' . $urlRegex . '$#'] = $handler;
    }

    public function get($url, $handler) {
        $this->aggiungiRotta('GET', $url, $handler);
    }

    public function post($url, $handler) {
        $this->aggiungiRotta('POST', $url, $handler);
    }

    public function put($url, $handler) {
        $this->aggiungiRotta('PUT', $url, $handler);
    }

    public function delete($url, $handler) {
        $this->aggiungiRotta('DELETE', $url, $handler);
    }

    public function instrada($url) {
        $method = $_SERVER['REQUEST_METHOD'];
        $url = parse_url($url, PHP_URL_PATH);

        if (isset($this->routes[$method])) {
            foreach ($this->routes[$method] as $route => $handler) {
                if (preg_match($route, $url, $matches)) {
                    $params = array_fi                    ng', ARRAY_FILTER_USE_KEY);
                    @var mixed $handler

                    $controller = new $handler[0]();
                    $action = $handler[1];

                    call_user_func_array([$controller, $action], $params);
                    return;
                }
            }
        }

        http_response_code(404);
        echo json_encode(['message' => 'Endpoint non trovato']);
    }
```

Backend

```php
public function leggiTutteAperte($id_utente_corrente = null)
{
    $query = "SELECT
                c.*,
                u.nome as nome_oste,
                u.cognome as cognome_oste,
                rp.stato AS stato_richiesta_utente
              FROM " . $this->table . " c
              JOIN utenti u ON c.id_oste = u.id
              LEFT JOIN richieste_partecipazione rp ON c.id = rp.id_cena AND rp.id_commensale = :id_utente_corrente
              WHERE c.stato = 'APERTA' AND c.dataOra > :current_time
              ORDER BY c.dataOra ASC";

    $current_time = (new \DateTime())->format('Y-m-d H:i:s');
    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(':id_utente_corrente', $id_utente_corrente);
    $stmt->bindParam( ':current_time', $current_time);
    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

```php
// --- Rotte Cene ---
$router->post('/api/cene', [DinnerController::class, 'crea']);
$router->get('/api/cene', [DinnerController::class, 'leggiTutte']);
```

```php
public function leggiTutte()
{
    $id_utente_corrente = null;
    $userData = \App\Core\AuthMiddleware::recuperaUtente();
    if ($userData) {
        $id_utente_corrente = $userData->id;
    }

    $dinnerModel = new Dinner();
    $dinners = $dinnerModel->leggiTutteAperte($id_utente_corrente);
    echo json_encode($dinners);
}
```

```javascript
function loadDinners() {
    $.ajax({
        url: `${API_BASE_URL}/cene`,
        method: 'GET',
        beforeSend: (xhr) => {
            const token = localStorage.getItem('jwtToken');
            if (token) {
                xhr.setRequestHeader('Authorization', 'Bearer ' + token);
            }
        },
        success: (dinners) => {
            const container = $('#dinners-container');
            container.empty();
            if (dinners.length === 0) {
                container.html('<p>Nessuna cena disponibile al momento.</p>');
                return;
            }
```

Alberatura e Design pattern

Quindi cosa succede quando viene fatta una chiamata da frontend (premendo un pulsante ad esempio)?

Progettazione Database

Il database è stato mappato per le sue tabelle 1:1 con le classi definite nel diagramma delle classi: utenti, cene, richieste_partecipazione, partecipazioni, recensioni, notifiche

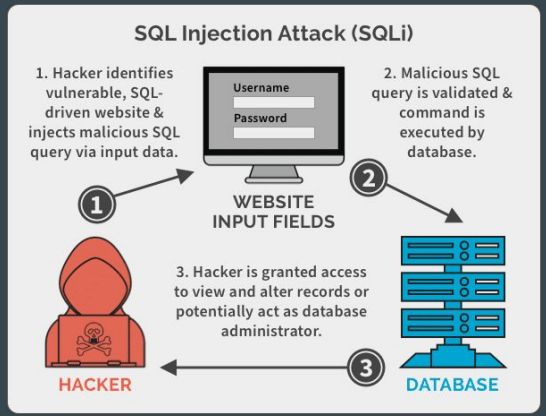
```sql
CREATE TABLE `partecipazioni` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `id_richiesta` INT NOT NULL UNIQUE,
  `id_cena` INT NOT NULL,
  `id_commensale` INT NOT NULL,
  `dataConferma` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  `statoPartecipante` ENUM('CONFERMATO', 'ANNULLATO_DA_UTENTE') DEFAULT 'CONFERMATO',
  FOREIGN KEY (`id_richiesta`) REFERENCES `richieste_partecipazione`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`id_cena`) REFERENCES `cene`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`id_commensale`) REFERENCES `utenti`(`id`) ON DELETE CASCADE
);

CREATE TABLE `recensioni` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `id_cena` INT NOT NULL,
  `id_valutatore` INT NOT NULL,
  `id_valutato` INT NOT NULL,
  `voto` INT NOT NULL CHECK (`voto` >= 1 AND `voto` <= 5),
  `commento` VARCHAR(255),
  `data` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (`id_cena`) REFERENCES `cene`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`id_valutatore`) REFERENCES `utenti`(`id`) ON DELETE CASCADE,
  FOREIGN KEY (`id_valutato`) REFERENCES `utenti`(`id`) ON DELETE CASCADE
);
```

Misure di Sicurezza

Autenticazione Stateless con JWT: autenticazione tramite JSON Web Token, il server non ha bisogno di mantenere uno stato di sessione, il Bearer Token viene trasmesso nell'header Authorization per proteggere gli endpoint.

Prevenzione di SQL Injection: Prepared Statement tramite PDO come misura fondamentale per prevenire attacchi di tipo SQL Injection, garantendo che gli input degli utenti non possano mai essere eseguiti come comandi sul database.

Autorizzazione basata sui ruoli: Lato controller, verifichi i permessi dell'utente (es. "solo l'oste di questa cena può annullarla"), garantendo che un utente non possa compiere azioni su dati che non gli appartengono.



Validazione e Evoluzione (Manutenzione e Test)

In questa fase si verifica se il software implementato soddisfa i requisiti e funzioni correttamente producendo il piano di test.

Piano di Test: il documento che definisce la strategia di testing. Include la progettazione dei casi di test (Unit, Integrazione, Sistema)

Unit Test (Test dei Componenti): Questi test sono eseguiti a livello di singola funzione o metodo

ID Test	Componente da Testare	Descrizione	Risultato Atteso
UT-01	User::crea()	Test sulla creazione di un utente con dati validi.	L'utente viene inserito correttamente nel database con la password "hashata".
UT-02	User::crea()	Test sulla creazione di un utente con una email già esistente.	Il metodo solleva un'eccezione o restituisce false, impedendo l'inserimento.
UT-03	Dinner::AggiornaPosti	Test sulla funzione di decremento posti disponibli	Il numero di posti disponibili per la cena diminuisce di 1.
UT-04	Dinner::AggiornaStato	Test sul cambio di stato da “aperta” a “completa”	Lo stato della cena nel database viene aggiornato correttamente.
UT-05	Participation::annulla()	Test sull'annullamento di una partecipazione.	Lo stato della partecipazione viene impostato su “annullato_da_utente”

Test di Integrazione

Questi test verificano l'interazione tra diversi moduli (es. Controller e Model) per validare i flussi funzionali.

ID Test	Caso d’uso	Descrizione	Dati di input	Risultato Atteso
UT-01	Iscrizione a cena	Test sulla creazione di un utente con dati validi.	Id utente valido, id cena valido, azione: “accettata”	Viene creata una richiesta_partecipazione a db con stato “in_attesa”. L’oste riceve una notifica
UT-02	Gestione iscrizioni	L’oste accetta una richiesta in attesa per una cena con un solo posto disponibile rimasto	Id richiesta valido	Il metodo solleva un'eccezione o restituisce false, impedendo l'inserimento.
UT-03	Annullamento partecipazione	Test sulla funzione di decremento posti disponibli	Id partecipazione valido	Lo stato della Partecipazione diventa “annullato_da_utente”
UT-04	Annullamento cena	Un Oste annulla una sua cena con 2 partecipanti	Id cena valido	Lo stato della cena diventa “annullata”, i 2 partecipanti ricevono una notifica di annullamento.
UT-05	Recensione Commensale-Oste	Un commensale che ha partecipato a una Cena lascia la recensione all’Oste	Cena, id oste, voto, commento	Lo stato della partecipazione viene impostato su “annullato_da_utente”

Validazione e Evoluzione (Manutenzione e Test)

In questa fase si verifica se il software implementato soddisfa i requisiti e funzioni correttamente producendo il piano di test.

Piano di Test: il documento che definisce la strategia di testing. Include la progettazione dei casi di test (Unit, Integrazione, Sistema)

Caso di Test: Creazione Cena

- Partizioni Equivalenti per "Numero Posti":
  - **Invalida:** Numeri ≤ 0 (es. -1, 0)
  - **Valida:** Numeri > 0 (es. 1, 4, 10)
- Partizioni per "Data":
  - **Invalida:** Data nel passato.
  - **Valida:** Data nel futuro.

Test di Sistema

Questi test simulano l'utilizzo reale dell'applicazione da parte degli utenti per validare l'intero sistema. Si basano sulla tecnica delle **partizioni in classi di equivalenza**.

ID Test	Descrizione	Dati di input	Risultato Atteso
UT-01	Creazione cena con dati validi	Titolo: "Cena Romana", Data: (domani), Posti: 4, altri campi compilati.	La cena viene creata e pubblicata correttamente e l'utente viene reindirizzato alla lista delle cene.
UT-02	Creazione cena con posti non validi	Titolo: "Cena Test", Data: (domani), Posti: 0, altri campi compilati.	Il metodo solleva un'eccezione o restituisce false, impedendo l'inserimento.

Test di Regressione

Ad ogni modifica del codice, una suite di test automatici (che include i principali Unit Test e Test di Integrazione) dovrà essere eseguita per assicurarsi che le nuove modifiche non abbiano introdotto difetti in funzionalità preesistenti.

Stress Test

Questi test verificano il comportamento del sistema sotto carico.

STRESS-01: Simulare l'iscrizione contemporanea di 10 utenti a una cena con soli 3 posti disponibili.

Risultato atteso: Il sistema deve accettare solo 3 richieste e non deve andare in crash. Lo stato della cena deve passare correttamente in “completata” dopo la terza accettazione da parte dell’oste.

Fine

Francesco Perrucci 17/09/2025