**Università degli Studi di Salerno**

Simone Cristiano 0512105070

Mario Offertucci 0512105184

Valeria Santarpia 0512105232

Smart Restaurant  
System Design Decomposition

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 19/11/2019 | 1.0 | Prima stesura del documento | Valeria Santarpia |
| 21/11/2019 | 1.1 | Analisi | Simone Cristiano Mario Offertucci Valeria Santarpia |
| 23/11/2019 | 1.2 | Aggiornamento in base alle analisi | Valeria Santarpia |
| 26/11/2019 | 1.3 | Revisione [Tutto l’aspetto visivo] | Simone Cristiano |
| 29/11/2019 | 1.4 | Definite le sezioni 2.5 e 2.6 | Valeria Santarpia |
| 03/11/2019 | 1.5 | Sistemato il layout del documento | Simone Cristiano |
| 17/12/2019 | 1.6 | Rifinite sezioni 2.1, 2.2 e 2.3 | Valeria Santarpia |

# 

# 1. Introduzione

**1*.*1 Scopo del sistema**

Lo scopo del sistema è di fornire un modo semplice per dirigere un ristorante, in particolare le comande.   
Infatti, grazie al sistema, la gestione delle comande risulta semplificata. Non c’è bisogno di portare il menù ai tavoli, poiché è già presente sul device posizionato sullo stesso e non si devono appuntare gli ordini e portare alla cucina poiché viene tutto gestito dal sistema.

**1*.*2 Obiettivi di progettazione**

**1*.*2.1 Criteri di prestazioni**

* **Throughput**Il sistema deve essere in grado di gestire un numero di richieste contemporanee pari al numero dei tavoli presenti in sala.
* **Memoria**Tutti i dati relativi alla struttura della sala, agli account e alle comande sono conservati in un database relazionale, quindi lo spazio occupato dal sistema dipende dalle dimensioni di questo database.

**1*.*2.2 Criteri di affidabilità**

* **Robustezza del sistema**L’inserimento dei dati, da parte dell’utente, è controllato in modo tale da evitare dati pericolosi.
* **Attendibilità**I risultati del sistema devono essere attendibili, ovvero i risultati devono rispettare le specifiche.
* **Disponibilità**Il sistema deve servire le richieste dei Tavoli quando il locale è aperto e degli altri tipi di utenti quando esso è attivo.
* **Sicurezza**Il sistema è progettato in modo tale che solo ed esclusivamente gli utenti autorizzati possono accedere ai dati.

**1*.*2.3 Criteri di costi**

* **Costi di sviluppo**Il sistema costerà 58.000 €.
* **Costo relativo all’installazione**Il sistema da realizzare ha un costo di 2.500 €.
* **Costo di manutenzione**Il costo della manutenzione è di 10.000 € annui.
* **Costo di amministrazione**Il gestore avrà il compito di amministrare il sistema, con un costo di 1.000 €.

**1*.*2.4 Criteri di manutenzione**

* **Estensibilità**Il sistema, essendo modulare, rende facile l’aggiunta di nuove funzionalità.
* **Modificabilità**Il sistema è stato progettato in modo da rendere agevole la modifica di funzioni esistenti.
* **Portabilità**Il sistema è stato progettato per essere fruito su un qualsiasi dispositivo Android.
* **Leggibilità**Il codice può essere compreso leggendo la documentazione allegata.
* **Tracciabilità dei requisiti**Grazie al RAD, si riescono ad individuare le funzioni relativi ai requisiti.

**1*.*2.5 Criteri end-user**

* **Usabilità**Il sistema supporta l’utente tramite un’interfaccia semplice e intuitiva che, in base al suo ruolo, gli fornisce una visione dedicata del sistema.

**1*.*3 Referenze**

* **Problem Statement:**   
  La descrizione del sistema.
* **RAD:**   
  Requisiti funzionali, non funzionali e casi d’uso.

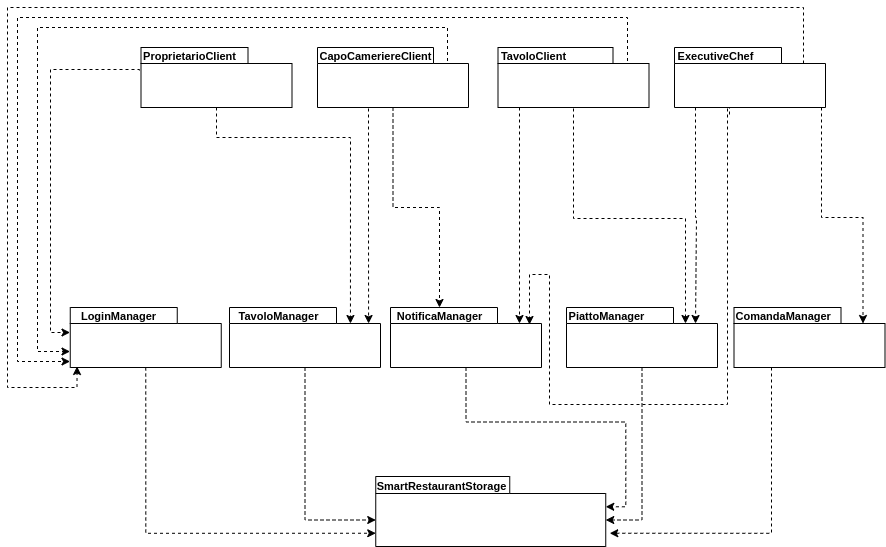
**1*.*4 Panoramica**

Nel secondo punto del documento viene presentata l’architettura del sistema proposto. In particolare, la decomposizione in sottosistemi, il mapping hardware/software, i dati persistenti, il controllo degli accessi e sicurezza, il controllo del flusso globale del sistema e le condizioni limite.  
Nel terzo punto vengono presentati i servizi dei sottosistemi.  
Nel quarto punto è illustrato un glossario dei termini.

# 2. Architettura di sistema proposta

**2*.*1 Decomposizione in sottosistemi**

Durante la fase di decomposizione di Smart Restaurant abbiamo diviso il sistema in piccoli sottosistemi con forte coesione. I sottosistemi, tra loro, sono debolmente accoppiati.  
Abbiamo tre gruppi di sottosistemi: interfaccia, logica applicativa e storage.  
Nell’interfaccia troviamo i vari Client di visualizzazione, separati in base al tipo di account, poiché ogni account può svolgere operazioni diverse e visualizzare pagine diverse.  
Nella logica applicativa troviamo i vari fornitori di servizi.  
Il LoginManager si occupa di gestire l’autenticazione e gli account del sistema.  
Il ComandeManager si occupa della gestione delle comande dei tavoli.  
Il PiattoManager gestisce la visualizzazione dei piatti ed il loro ordine.  
Il TavoloManager offre le funzionalità per la gestione dei tavoli.  
Il NotificaManager gestisce l’invio e la ricezione delle notifiche scambiate tra i vari utenti del sistema.  
Lo storage dei dati è gestito da SmartRestaurantStorage, dove vengono memorizzati i dati persistenti del sistema.  
Il seguente diagramma UML mostra un overview dei sottosistemi e delle loro relazioni.



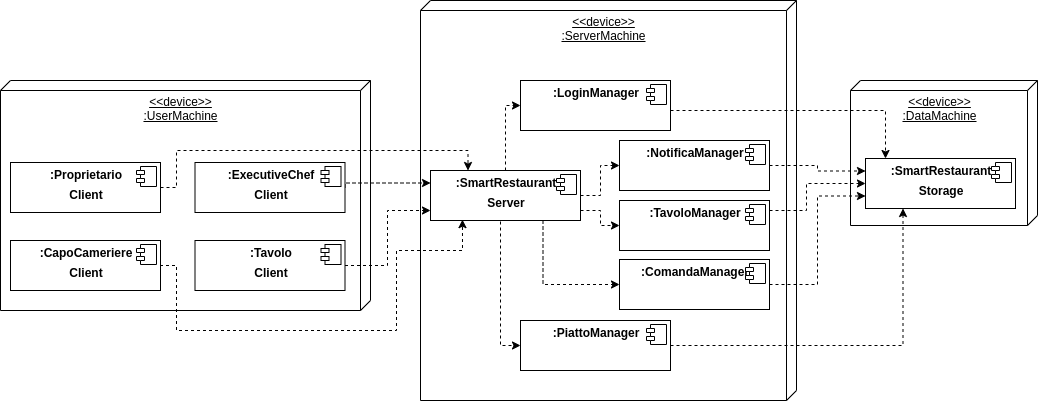
**2*.*2 Mapping hardware/software**

Smart Restaurant è un sistema distribuito al quale l’utente può accedere usando un device Android. Quindi, esso verrà utilizzato da diversi utenti su diverse macchine, presenti tutte nello stesso locale.  
Possiamo distinguere tre tipi di nodi:

* UserMachine, che fornisce l’interfaccia utente;
* ServerMachine, dove viene eseguita la logica applicativa del sistema;
* DataMachine, dove avviene lo storage dei dati.

ProprietarioClient, ExecutiveChefClient, CapoCameriereClient e TavoloClient vengono eseguite nella UserMachine. In base ai dati forniti durante la fase di login verrà scelto il tipo di client.  
Tutti i sottosistemi riguardanti la logica applicativa sono collocati su una singola ServerMachine.  
Per SmartRestaurantStorage viene usata una macchina dedicata, la DataMachine.

Per la realizzazione dell’organizzazione del sistema abbiamo scelto di usare un server scritto in linguaggio Java che utilizza Dispacher e Queue per fornire i servizi.   
Per l’implementazione dell’interfaccia client abbiamo scelto di utilizzare il linguaggio Java per Android, realizzando un’applicazione event-based. In questo modo rendiamo semplice l’utilizzo del sistema, poiché risulta intuitivo l’utilizzo di un’app su device Android, e sfruttiamo le potenzialità forniteci dal dispositivo.   
Di seguito è mostrato il mapping hardware/software con un UML deployment diagram.



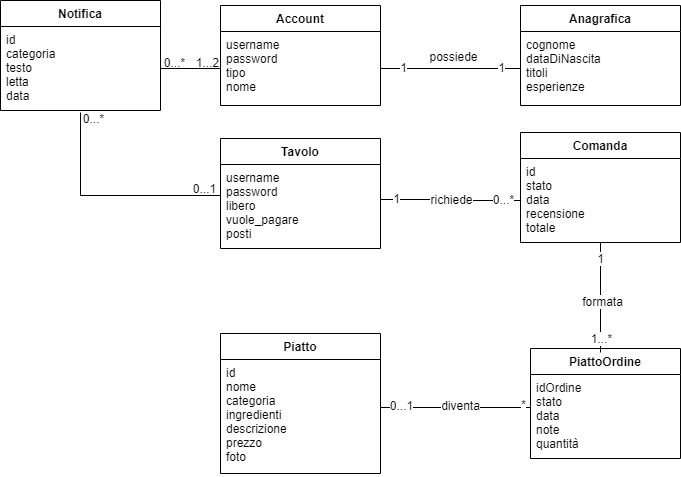
**2*.*3 Gestione delle persistenze**

I dati del sistema che devono essere resi permanenti sono:

* Informazioni degli account (proprietario, executive chef, capo cameriere e tavolo);
* Informazioni del menu;
* Informazioni relative alle comande;
* Informazioni riguardanti le notifiche scambiate tra executive chef, tavolo e capo cameriere.

Per gestire tali persistenze è stato scelto l’utilizzo di un DBMS relazionale, in specifico MySQL.  
Questa scelta garantisce una semplice manipolazione dei dati e totale trasparenza da parte di chi lo utilizza.  
Per rendere più semplice la manipolazione dei dati i tipi di account sono stati specializzati in Account e Tavolo.  
Account viene usato per Proprietario, Executive Chef e Capo Cameriere, poiché mantengono le stesse informazioni.  
Tavolo, invece, non contiene l’anagrafica ma le informazioni sullo stato del tavolo, il numero di posti e le comande legate ad esso.

Lo schema del database, espresso in UML, è riportato in basso.



**2*.*4 Controllo degli accessi e sicurezza**

Il controllo degli accessi e della sicurezza è illustrato dalla seguente matrice degli accessi.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Oggetti**  **Attori** | **Tavolo** | **Menu** | **PiattoOrdinato** | **Comanda** | **Notifica** |
| **Proprietario** | crea  visualizza  modifica  rimuovi |  |  |  |  |
| **Executive Chef** |  | visualizza | visualizza  check  rimuovi | visualizza  check | crea |
| **Capo Cameriere** | visualizza  modifica |  | visualizza | visualizza  modifica | visualizza |
| **Tavolo** | visualizza | visualizza | crea  visualizza  modifica  rimuovi | crea  visualizza  modifica | crea |

**2*.*5 Controllo globale del software**

Il controllo globale del software è di tipo event-based.   
Gli eventi guidano l’esecuzione del sistema. Essi sono gestiti tramite Dispacher e Queue. Gli observers interessati ad un evento effettuano una subscrive alla relativa coda ed attendono di ricevere una notifica ogni qual volta che questo si verifica. Quando l’evento viene lanciato, il Dispacher si occupa di notificarlo a chi è interessato. Tali eventi sono trattati in modo sincrono.  
Eventuali concorrenze potrebbero verificarsi su PiattoOrdinato e Comanda, che possono essere visualizzati e/o modificati da Executive Chef, Capo Cameriere e Tavolo. Per risolvere il problema abbiamo reso l’accesso a tali oggetti sincronizzato, in modo da evitare incoerenze.  
Tale PiattoOrdinato può anche essere rimosso da Executive Chef o Tavolo. La rimozione, da parte del Tavolo, può avvenire solo in fase di creazione della comanda e non è possibile una volta che la stessa è stata confermata ed inviata alla cucina (se il Tavolo non desidera più tale piatto dovrà riferirlo ad un cameriere che provvederà a comunicare la modifica all’Executive Chef). L’Executive Chef potrà rimuovere il piatto dalla lista dei piatti da preparare senza provocare problemi di concorrenza data la natura sincronizzata dell’oggetto.  
Su ogni device, l’esecuzione del sistema è sincrona e non prevede l’uso di threads.

**2*.*6 Condizioni limite**

Le boundary condition di Smart Restaurant riguardano l’installazione, l’avvio e la sospensione del sistema.

* **Installazione**Il sistema è installato da un sistemista e da un programmatore, che provvederanno, inoltre, alla creazione degli account di Proprietario, Executive Chef e Capo Cameriere, della sala e del menu.   
  Tutti gli altri oggetti vengono creati dai vari utenti usando il sistema.
* **Avvio del Server**Il Server può essere avviato dal Proprietario, dall’Executive Chef o dal Capo Cameriere. Dopo il suo avvio, viene attivato il DBMS MySQL. Viene effettuata la connessione al DBMS tramite il driver JDBC. Successivamente viene avviata la JVM sul Server e vengono letti i dati di configurazione del database. Infine, vengono caricati tutti i dati in memoria volatile. Il sistema è pronto all'uso.
* **Sospensione del Server**Il Server viene spento quando il locale non è in attività. In tali momenti possono essere effettuate manutenzioni o aggiornamenti.
* **Fallimento del Server**Il Server può fallire in caso di interruzione inaspettata dell'alimentazione e/o di guasti hardware. Il Server può andare in crash quando viene sollevata un'eccezione che non è stata gestita.
* **Avvio del Client**Il Client viene avviato all’accensione dei dispositivi.
* **Sospensione del Client**Il Client viene sospeso con lo spegnimento dei dispositivi.

|  |  |  |
| --- | --- | --- |
| Id | SY01 | |
| Nome | Avvio del Server | |
| Attore iniziale | Proprietario, Executive Chef o Capo Cameriere | |
| Precondizioni | Il Server è spento | |
| Flusso degli eventi | ***Attore*** | ***Sistema*** |
| 1. L’attore accede alla macchina Server e clicca su ‘Start’ |  |
|  | 2. Il sistema riceve la richiesta ed avvia tutti i servizi. |
| Condizione di uscita | Il Server è attivo e pronto a fornire servizi | |

|  |  |  |
| --- | --- | --- |
| Id | SY02 | |
| Nome | Sospensione del Server | |
| Attore iniziale | Proprietario, Executive Chef o Capo Cameriere | |
| Precondizioni | Il Server è attivo | |
| Flusso degli eventi | ***Attore*** | ***Sistema*** |
| 1. L’attore accede alla macchina Server e clicca su ‘Stop’ |  |
|  | 2. Il sistema riceve la richiesta, termina le operazioni in corso, effettua i dovuti salvataggi e si sospende |
| Condizione di uscita | Il Server è sospeso | |

|  |  |  |
| --- | --- | --- |
| Id | SY03 | |
| Nome | Fallimento del Server | |
| Attore iniziale | Proprietario, Executive Chef o Capo Cameriere | |
| Precondizioni | Il Server è attivo | |
| Flusso degli eventi | ***Attore*** | ***Sistema*** |
|  | 1. Il Server mostra al Client una pagina di errore |
| 2. L’attore si accorge della presenza di un problema sul Server e decide di riavviarlo |  |
|  | 3. Il Server salva i dati presenti in memoria volatile e si riavvia riattivando tutti i servizi |
| Condizione di uscita | Il Server è nuovamente in funzione | |

# 3. Servizi del sottosistema

Di seguito verranno descritti I servizi offerti dai vari sottosistemi, con relativa descrizione.

**- LoginManager**

|  |  |
| --- | --- |
| **Servizi** | **Descrizione** |
| CheckLogin | Consente di accedere al proprio account |

**- ComandaManager**

|  |  |
| --- | --- |
| **Servizi** | **Descrizione** |
| SegnaComeFatto | Spunta un piatto come preparato |
| SegnaComeNonFatto | Rimuove un piatto dai piatti da preparare |
| RequestAllComande | Richiede tutte le comande ancora non completate |

**- PiattoManager**

|  |  |
| --- | --- |
| **Servizi** | **Descrizione** |
| RequestPiatti | Fornisce i piatti del menu di una specifica categoria (primo, secondo, …) |
| SendOrdine | Invia la richiesta, da parte di un Tavolo, della preparazione di uno o più piatti del menu |

**- TavoloManager**

|  |  |
| --- | --- |
| **Servizi** | **Descrizione** |
| RequestAllTavoli | Fornisce tutti i tavoli presenti in sala |
| RequestTavoliLiberi | Fornisce tutti i tavoli liberi presenti in sala |
| RequestTavoliOccupati | Fornisce tutti i tavoli occupati presenti in sala |
| RequestTavolo | Fornisce un tavolo specifico |
| SendNewTavolo | Crea e memorizza un nuovo tavolo per la sala |
| SendChangeStateTavolo | Modifica lo stato di un tavolo, rendendolo libero o occupato |
| SendChangePayTavolo | Modifica lo stato del pagamento di un tavolo |
| SendRemove | Rimuove un tavolo dalla sala |

**- NotificaManager**

|  |  |
| --- | --- |
| **Servizi** | **Descrizione** |
| RequestNotifiche | Fornisce tutte le notifiche per un determinato destinatario |
| SendNotifica | Invia una nuova notifica |

# 4. Glossario

|  |  |
| --- | --- |
| **Termine** | **Definizione** |
| **Android** | Sistema operativo per dispositivi mobili. |
| **Client** | Un qualunque componente che accede ai servizi o alle risorse di un'altra componente, detta server. |
| **Coesione** | Misura di quanto strettamente correlate siano le varie funzionalità messe a disposizione da un singolo modulo. |
| **Comanda** | Nel gergo dei camerieri indica un’ordinazione. |
| **Crash** | Blocco o terminazione improvvisa del sistema. |
| **Device** | Nel linguaggio dell'informatica indica un dispositivo elettronico. |
| **Event-based** | Paradigma di programmazione. |
| **Nodo** | Dispositivo del sistema. |
| **Overview** | Veduta d’insieme del sistema. |
| **Server** | Componente o sottosistema informatico di elaborazione e gestione del traffico di informazioni che fornisce, a livello logico e fisico, un qualunque tipo di servizio ad altre componenti. |
| **Storage** | Supporto per la memorizzazione. |
| **Thread** | Suddivisione di un processo in più sottoprocessi. |