Documentazione - TokyoGroup BarberShop

Membri del gruppo:

- Daniele Lazzari, 1080979
- Mattia Curto, 1081143
- Andrea Milani, 1079678
- Andrea Roggeri, 1079033

Indice

1. Progetto

Il progetto consiste nella creazione di un'applicazione completa per la gestione degli appuntamenti presso un barbiere. L'applicazione è strutturata per soddisfare le necessità sia dei gestori del salone sia dei clienti.

Funzionalità principali:

- **Gestore:**
  - Creazione, modifica e cancellazione di appuntamenti.
  - Gestione dei messaggi inviati dai clienti.
  - Modifica dei servizi offerti (aggiunta/rimozione di servizi).
  - Modifica delle credenziali di accesso e logout sicuro.
- **Cliente:**
  - Prenotazione e cancellazione di appuntamenti.
  - Invio di messaggi al gestore.
  - Modifica delle credenziali di accesso.
  - Eliminazione dell'account con conferma di sicurezza.

Obiettivi:

1. Migliorare l'efficienza della gestione degli appuntamenti.
2. Ridurre errori e malintesi tramite un sistema centralizzato.
3. Fornire una piattaforma intuitiva per l'uso quotidiano.

2. Software Engineering Management (PROJECT PLAN)

Il project plan si trova al seguente link:
https://github.com/a-roggeri/SWE-Project/blob/a35c85752cdba740fe74fc6d474185df0fd3b3d6/Project%20Plan%20(1).docx

3. Software Life Cycle

Il ciclo di vita del software è basato sul modello a cascata sviluppando in ordine analisi dei requisiti, progettazione, implementazione, verifica e manutenzione.

Fasi del ciclo di vita:

1. **Analisi dei requisiti:** basate sul modello MoSCoW .
2. **Progettazione:** creazione di diagrammi UML e definizione dell'architettura.
3. **Sviluppo:** implementazione incrementale di funzionalità.
4. **Testing:** test effettuati sulla logica del programma.
5. **Verifica:** utilizzo analisi dinamica sui test.
6. **Manutenzione:** manutenzione preventiva e adattiva.

L'organizzazione del team è basata sui principi SWAT, l'unione e la collaborazione sono stati fondamentali per raggiungere gli obiettivi.
Le riunioni avvenivano principalmente su google meet. Durante le fasi di analisi dei requisiti e di progettazione erano molto più lunghe e strutturate, durante le fasi successive erano riunioni di allineamento molto più brevi e snelle.

Abbiamo scelto di utilizzare il modello Waterfall, un approccio strutturato e sequenziale allo sviluppo che ci ha permesso di procedere attraverso fasi ben delineate. Questo modello include l'analisi dei requisiti, la progettazione del sistema e del software, l'implementazione, il testing e infine la manutenzione. La decisione di adottare il modello Waterfall è stata motivata dalla natura ben definita e stabile dei requisiti del progetto, che ha reso possibile una pianificazione dettagliata e accurata delle attività.

Descrizione delle fasi:

- Analisi dei requisiti: In questa fase iniziale abbiamo cercato di comprendere le specifiche esigenze della gestione degli appuntamenti ed identificato requisiti cruciali come la prenotazione, la cancellazione e la modifica degli appuntamenti e registrazione dell'utente sia come cliente che come gestore. È stato essenziale definire chiaramente questi requisiti per assicurare che il software soddisfacesse tutte le funzionalità necessarie.
- Progettazione: Con i requisiti ben definiti, abbiamo progettato l'architettura del sistema. Questa fase ha incluso la creazione di diagrammi UML che delineano la struttura delle varie parti del software. Abbiamo anche definito le interfacce utente che permettono una facile interazione sia per il parrucchiere che per i clienti.
- Implementazione: Durante l'implementazione, abbiamo trasformato le specifiche tecniche in un'applicazione funzionante. Ogni modulo è stato sviluppato in modo da integrarsi con gli altri, formando un sistema coeso.

- Testing: Questa fase è stata fondamentale per garantire la qualità e l'efficacia del software. Abbiamo condotto diversi tipi di test, compresi i test funzionali per verificare che tutte le funzioni rispondessero ai requisiti specificati. Questo ha permesso di identificare e correggere eventuali difetti prima della consegna.
- Manutenzione: La fase finale ha incluso il monitoraggio continuo del sistema per risolvere problemi emergenti e aggiornare il software con eventuali funzionalità richieste o suggerite degli utenti.

In ogni fase, abbiamo potuto concentrarci su specifici obiettivi senza sovrapposizioni, con il vantaggio di ridurre le ambiguità e migliorare la gestione delle risorse.

Questo modello ci ha anche permesso di mantenere una documentazione rigorosa e completa, facilitando la fase di manutenzione. Nonostante la limitata flessibilità del modello Waterfall, che rende complesse le modifiche ai requisiti una volta che il processo è avviato, la sua prevedibilità e la struttura lineare hanno contribuito significativamente al successo del progetto in un contesto dove i requisiti erano chiari.

4. Configuration Management

Il repository è strutturato nel seguente modo:

- branches:
    - *main*: Contiene le versione definitiva del codice
- cartelle:
    - *GestioneAppuntamenti*, che contiene:
        - Il codice sorgente del progetto ed i relativi test.
        - Le varie risorse utilizzate, come database e immagini.
        - I file di configurazione del progetto maven.
    - *UML*: Contiene tutti i diagrammi UML:
        - in formato .mdj per StarUML
        - in formato .xmi per Papyrus
        - formato jpeg
    - *Documentazione*: contiene la documentazione del progetto.

Tutto il materiale è salvato nella repository di GitHub.

5. People Management and Team Organization

Il team è stato organizzato per massimizzare le competenze e ottimizzare la collaborazione. Ogni componente ha partecipato ad ogni attività. La suddivisione del lavoro è stata concordata tra i membri del gruppo sulla base di necessità personali e lavorative. Abbiamo comunque suddiviso la gestione in ruoli nel seguente modo.

Ruoli assegnati:

- **Daniele Lazzari:** backend, frontend, test, documentazione
- **Mattia Curto:** backend, frontend, UML, documentazione
- **Andrea Milani:** backend, frontend, UML, documentazione
- **Andrea Roggeri:** backend, frontend, progettista database, test, documentazione

6. Software Quality

Il team si è prefissato di sviluppare un'applicazione che rispetti rigorosi parametri di qualità, seguendo i requisiti tassonomici di McCall.Gli attributi di qualità sono stati suddivisi in tre categorie principali: produzione, revisione e transizione del codice. Di seguito illustriamo le qualità che descrivono il nostro sistema.

Produzione del Codice e Utilizzo del Software

Durante la produzione del codice e il successivo utilizzo da parte degli utenti finali, il software soddisfa i seguenti requisiti di qualità:

- **Correttezza**: L'applicazione soddisfa pienamente i requisiti specificati ed è stata sviluppata in conformità alle specifiche, garantendo il corretto funzionamento delle sue funzionalità.
- **Affidabilità**: Il software è stato sottoposto a test approfonditi per minimizzare la presenza di bug. È in grado di svolgere le funzioni richieste in modo accurato e senza errori.
- **Efficienza**: Non richiede risorse hardware particolarmente avanzate per il funzionamento. Il software è stato ottimizzato per garantire prestazioni elevate anche su sistemi con requisiti minimi.
- **Integrità**: L'accesso al sistema è garantito esclusivamente agli utenti autorizzati che hanno effettuato correttamente la registrazione e sono inclusi nel database.
- **Usabilità**: L'interfaccia grafica è stata progettata per essere semplice e intuitiva, in modo che il software possa essere utilizzato senza particolari competenze tecniche.

Manutenibilità e Interazione con l'Ambiente

Il team si è impegnato a garantire che il software sia manutenibile e facilmente integrabile in ambienti diversi. Sono state rispettate le seguenti qualità:

- **Testabilità**: Le funzionalità sono testabili in qualsiasi momento mediante test automatici implementati con JUnit.

- **Manutenibilità**: Il codice è stato strutturato in modo chiaro e modulare, separando le classi di logica, modello e interfaccia. Questa suddivisione facilita l'individuazione e la risoluzione di eventuali errori.
- **Flessibilità**: Il sistema è predisposto per l'implementazione di nuove funzionalità senza modificare significativamente l'architettura esistente.

Transizione del Codice

Per quanto riguarda la transizione del codice e l'adattabilità a nuovi ambienti, il software garantisce le seguenti qualità:

- **Portabilità**: L'applicazione è compatibile con i principali sistemi operativi desktop, tra cui Windows, Linux e macOS.
- **Riusabilità**: La progettazione modulare del codice consente di riutilizzare componenti e moduli in altri contesti applicativi.

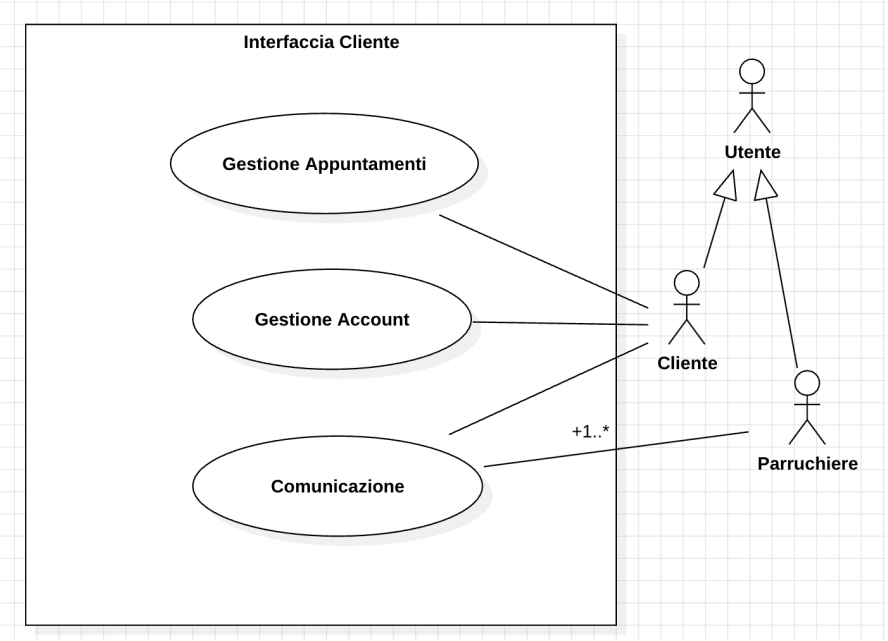
Stabilità e Tolleranza ai Guasti

Un'attenzione particolare è stata dedicata alla stabilità del sistema e alla sua tolleranza ai guasti. Durante i test, sono stati introdotti controlli rigorosi per garantire che il sistema rimanga stabile, anche in caso di input errati o azioni non corrette da parte dell'utente.

7. Requirements Engineering

I requisiti del progetto sono stati definiti durante la fase di elicitazione, in cui abbiamo utilizzato linguaggi formali e semi-formali per descriverli. Nello specifico, abbiamo adottato una descrizione in linguaggio naturale per una maggiore comprensibilità e integrato l'uso di diagrammi UML, come gli **Use Case Diagram**, per rappresentare visivamente come gli utenti interagiscono con il sistema. Questo approccio ci ha permesso di analizzare e comprendere meglio le esigenze e i comportamenti previsti degli utenti.

Ad esempio, il seguente diagramma dei casi d'uso raffigura gli utenti e ciò che vogliono realizzare nel sistema 'interfaccia cliente'.



A partire dai requisiti elicitati, abbiamo redatto la **Specifica dei Requisiti**, la quale ha costituito la base per l'attività di validazione. Tale attività è stata fondamentale per garantire che il sistema fosse sviluppato in modo conforme alle aspettative e alle specifiche concordate.

Per la gestione e la prioritizzazione dei requisiti, abbiamo adottato il modello **MoSCoW**, che ci ha consentito di organizzare i requisiti in base alla loro importanza e urgenza:

- **Must**: Requisiti essenziali che devono essere implementati per il funzionamento del sistema.
- **Should**: Requisiti importanti, ma non critici, che sarebbe preferibile includere.
- **Could**: Requisiti desiderabili che possono migliorare l'esperienza utente, ma non sono indispensabili.
- **Won't**: Requisiti che, al momento, possono essere posticipati o esclusi, in base alle priorità del progetto.

Il modello **MoSCoW** ci ha fornito la flessibilità necessaria per adattarci alle esigenze in evoluzione del progetto, concentrandoci sulle funzionalità più critiche e significative per il successo complessivo del sistema. Grazie a questa metodologia, siamo stati in grado di bilanciare le esigenze degli utenti con le risorse e i tempi disponibili, garantendo una pianificazione efficace e mirata.

Di seguito sono riportati i requisiti che costituiscono i Must Have del progetto, nel dettaglio ed i requisiti di Should Have che sono stati implementati nel tempo a disposizione:

MUST HAVE	SHOULD HAVE	COULD HAVE	WON'T
Gestione appuntamenti	Cambio password	Recensioni su gestore	Notifiche
Invio messaggi	Eliminazione account		Interfaccia web
Login e registrazione	Modifica servizi offerti		
Report settimanali			

7.1 Specifica dei Requisiti

7.1.1 Schermata Iniziale e Accesso

La schermata iniziale permette l'accesso al sistema attraverso due modalità: **Login** o **Registrazione**.
Un cliente che non è loggato può registrarsi fornendo le seguenti informazioni:

- Nome Utente
- Password
- Tipologia

Questi dati vengono salvati nel database e verificati durante ogni tentativo di login.
L'accesso al sistema è consentito solo se:

1. Nome Utente inserito è presente nel database.
2. La password corrisponde a quella associata al Nome Utente.

Nel caso di accesso non valido comparirà un messaggio di errore.
Gli utenti registrati possono accedere al sistema come **Clienti**, mentre gli amministratori accedono come **Gestori**, avendo accesso a funzionalità avanzate.

7.1.2 Menu Principale

Dopo l'accesso, viene mostrata la schermata principale, che offre diverse opzioni in base al ruolo dell'utente:

- **Cliente**:
    - Prenotare appuntamenti con un parrucchiere.
    - Visualizzare e gestire appuntamenti già prenotati.
    - Modificare i dati del proprio account o eliminarlo.
    - Inviare messaggi al gestore per comunicazioni.
- **Gestore**:
    - Visualizzare il calendario settimanale degli appuntamenti.
    - Creare, modificare o cancellare appuntamenti.
    - Gestire i servizi offerti e il personale.
    - Rispondere ai messaggi dei clienti.
    - Modificare i dati dell’account.

Il menu principale è progettato per essere intuitivo e accessibile, con un layout che facilita la navigazione tra le funzionalità.

7.1.3 Prenotazione di un Appuntamento

La funzionalità di prenotazione di un appuntamento prevede i seguenti passi:

1. Il cliente seleziona il **parrucchiere**, il **servizio desiderato** e l'**orario disponibile**.
2. Una volta effettuata la scelta, l'appuntamento viene salvato nel database e confermato con un messaggio all'utente.
3. Il cliente può modificare o cancellare l'appuntamento fino a 24 ore prima dell'orario prenotato.

In caso di errori o conflitti (es. appuntamenti già prenotati nello stesso orario), il sistema avvisa l'utente e propone alternative disponibili.

7.1.4 Gestione degli Appuntamenti (Gestore)

Il gestore può:

- Visualizzare gli appuntamenti settimanali in una visualizzazione calendario.
- Creare nuovi appuntamenti per i clienti, selezionando:
    - Nome del cliente
    - Servizio richiesto
    - Data e orario
- Modificare o cancellare appuntamenti esistenti.
- Gestire le richieste di modifica inviate dai clienti.

Le modifiche vengono propagate automaticamente nel database, assicurando coerenza e aggiornamenti in tempo reale.

7.1.5 Sicurezza e Gestione degli Utenti

- Solo gli utenti registrati possono accedere al sistema, garantendo un controllo sugli accessi.
- Gli utenti possono modificare la propria password tramite un'apposita funzionalità.

7.2 MoSCoW

I requisiti specificati nel punto 7.1 rappresentano quelli che abbiamo identificato come fondamentali per il successo del nostro sistema, classificati come **Must Have**. Seguendo il metodo MoSCoW, abbiamo inoltre etichettato altri requisiti come importanti ma non indispensabili (**Should Have**), desiderabili ma secondari (**Could Have**), e non implementabili nella versione attuale (**Won't**).

7.2.1 Must Have

I requisiti essenziali, che costituiscono la base per il corretto funzionamento del sistema e senza i quali non sarebbe possibile raggiungere gli obiettivi principali, sono:

- Gestione appuntamenti
- Invio messaggi
- Login e registrazione
- Report settimanali

7.2.2 Should Have

I requisiti ritenuti importanti, ma che possono essere implementati in un secondo momento se il tempo e le risorse lo permettono, includono:

- Cambio password
- Eliminazione account
- Modifica servizi offerti

7.2.3 Could Have

I requisiti secondari e desiderabili, che non influiscono sul funzionamento principale del sistema ma possono essere inclusi in versioni future per migliorare l'esperienza utente, sono:

- Recensioni su gestore

7.2.4 Won't

I requisiti che non verranno implementati in questa versione del sistema, poiché sono stati giudicati non prioritari o fuori ambito, includono:

- Notifiche

- Interfaccia web

7.3 KANO

Abbiamo utilizzato il modello KANO per analizzare e classificare le esigenze e le aspettative dal punto di vista dei clienti. Potrebbe diventare molto utile per capire quali funzionalità si potrebbero aggiungere al nostro software per soddisfare al massimo i clienti.

Il modello Kano identifica tre categorie principali di requisiti:

- Requisiti must have: Requisiti essenziali che i clienti si aspettano di trovare, la mancanza di essi possono portare il cliente ad abbandonare l'uso del software. All'interno pensiamo che ci debba essere un login ed una interfaccia semplice ed intuitiva per la gestione dell'appuntamento per quanto riguarda il cliente ed una visualizzazione immediata degli appuntamenti che deve effettuare ogni gestore.
- Requisiti unidimensionali: Requisiti che aumentano la soddisfazione del cliente. Maggiore è il livello di soddisfazione di questi requisiti, maggiore sarà la soddisfazione complessiva del cliente. In questa categoria facciamo rientrare l'invio dei messaggi da parte del cliente al gestore ed il selezionamento del parrucchiere che si preferisce.
- Requisiti attraenti: Requisiti che sorprendono i clienti, anche se non sono necessari. Tendono a differenziare il prodotto dalla concorrenza ed aggiungono un valore al prodotto. Tra questi troviamo i report settimanali del gestore che è in grado di vedere il fatturato e per il cliente il selezionamento di servizi specifici.

8. Modelling

La modellazione è stata effettuata utilizzando UML per rappresentare in modo chiaro la struttura e il comportamento del sistema.

Diagrammi realizzati:

- diagramma di sequenza
- diagramma delle attività
- diagramma di casi d'uso
- diagramma di comunicazione
- diagramma dei pacchetti
- diagramma classi
- diagramma dei componenti
- state Machine

8.1 Diagramma di Sequenza - Gestione Appuntamenti

Il diagramma di sequenza mostra il flusso delle interazioni tra i vari componenti del sistema per le principali funzionalità di gestione degli appuntamenti. Questo diagramma è stato fondamentale per modellare il comportamento del sistema e garantire la corretta gestione delle operazioni.

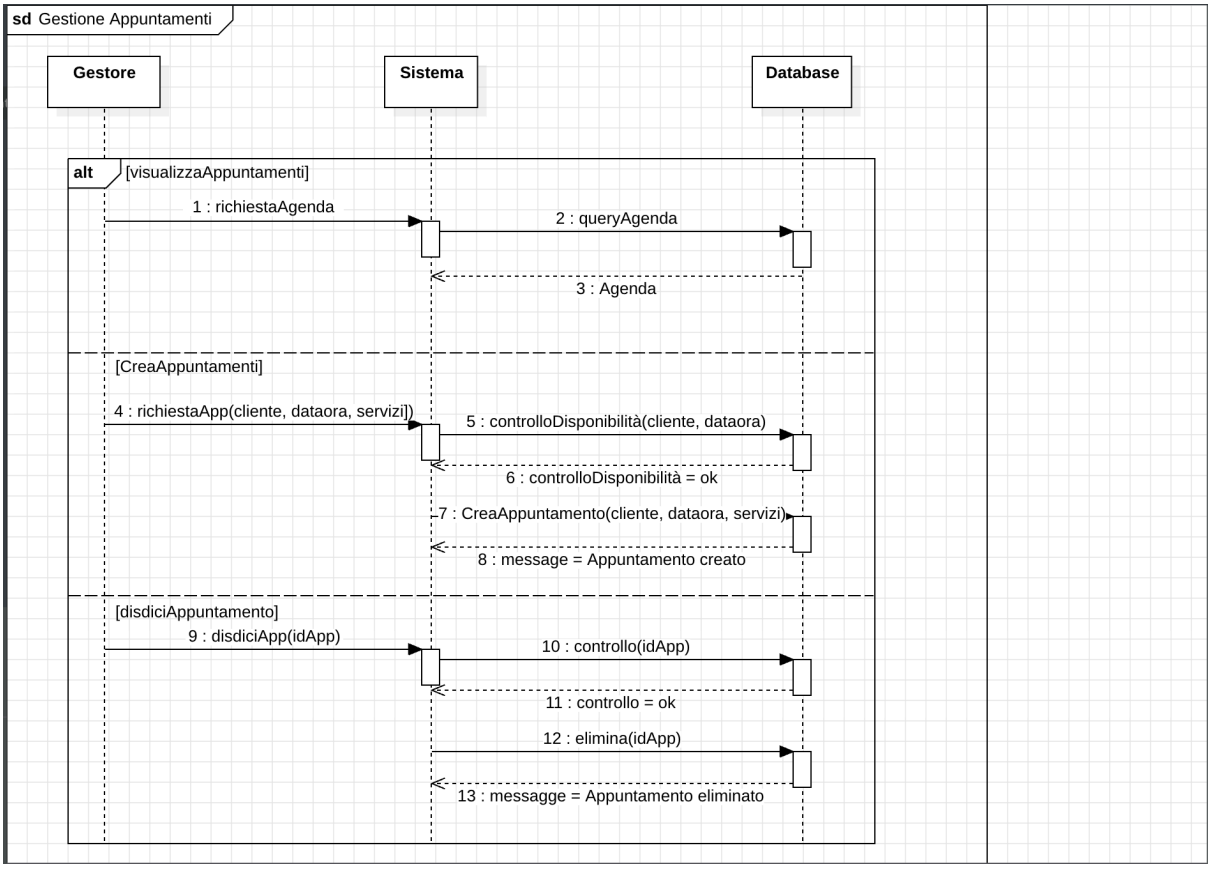
Abbiamo utilizzato il diagramma di sequenza per rappresentare tre casi principali:

1. Visualizzazione Appuntamenti: Il gestore richiede l'agenda degli appuntamenti, il sistema interroga il database e restituisce l'elenco aggiornato degli appuntamenti.
2. Creazione Appuntamenti: Il gestore invia una richiesta per creare un appuntamento, specificando il cliente, la data, l'orario e il servizio richiesto. Il sistema verifica la disponibilità, aggiorna il database e conferma l'avvenuta creazione.
3. Cancellazione Appuntamenti: Il gestore richiede di cancellare un appuntamento specifico tramite il relativo ID. Il sistema verifica l'esistenza dell'appuntamento, lo elimina dal database e invia una conferma.

Le interazioni principali tra gli attori (Gestore, Sistema e Database) includono:

- Gestione della disponibilità: Il sistema controlla la disponibilità del cliente e dello slot temporale prima di confermare l'appuntamento.
- Notifiche: Il sistema invia messaggi di conferma o errore per garantire una comunicazione chiara con il gestore.
- Aggiornamento del database: Tutte le operazioni modificano il database in tempo reale per mantenere coerenza e aggiornamenti costanti.

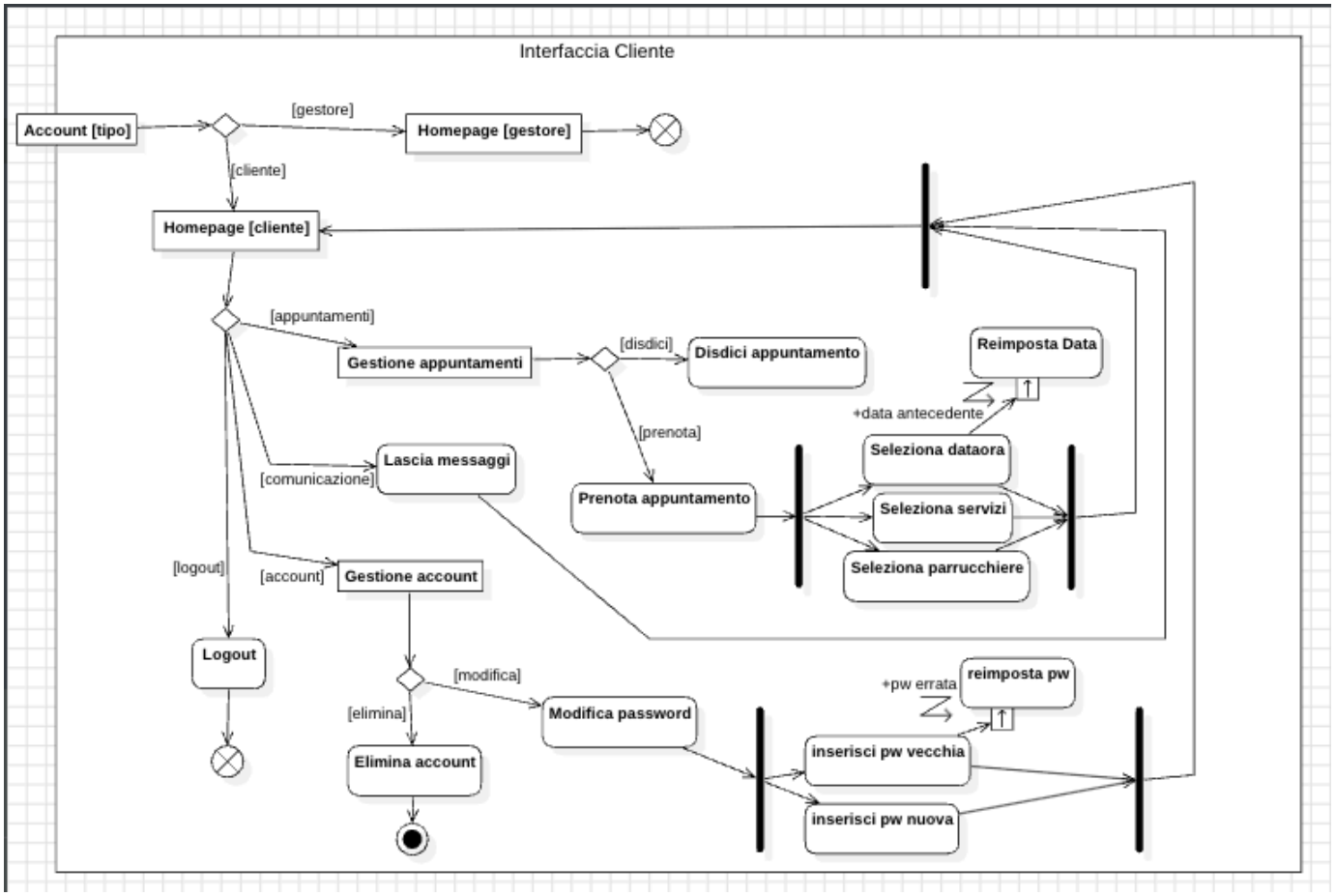
Questo diagramma si è evoluto durante lo sviluppo del sistema, integrando ulteriori dettagli per gestire eccezioni e garantire una maggiore stabilità nelle operazioni di accesso e aggiornamento dei dati.



8.2 Diagramma delle Attività

Il diagramma rappresenta un flusso di attività per un'interfaccia cliente, relativo a un sistema di gestione di appuntamenti o servizi. I principali dettagli del diagramma sono:

1. Account e Homepage:
    - Si parte dall'utente che accede al sistema tramite il proprio Account. A seconda del tipo di account ([tipo]), l'utente viene indirizzato alla Homepage [cliente] o alla Homepage [gestore].
    - Nel caso del gestore, il flusso termina subito dopo l'accesso alla sua homepage siccome è stato fatto il diagramma per l'interfaccia cliente.
2. Gestione appuntamenti:
    - Dalla Homepage del cliente, l'utente può accedere alla gestione degli appuntamenti, dove può:
        - Disdire un appuntamento, se già programmato.
        - Prenotare un appuntamento, procedendo a:
            - Selezionare data e ora (Seleziona data ora).
            - Scegliere i servizi richiesti (Seleziona servizi).
            - Selezionare il parrucchiere (Seleziona parrucchiere).
        - Se necessario, l'utente può impostare la nuova data, ma solo se la data scelta è antecedente a quella attuale.
3. Gestione account:
    - Gli utenti possono gestire il proprio account scegliendo l'opzione Gestione account:
        - Modifica password: permette all'utente di aggiornare la propria password inserendo quella vecchia (inserisci pw vecchia) e successivamente quella nuova (inserisci pw nuova). In caso di errore nella vecchia password, è possibile reimpostarla (reimposta pw).
        - Elimina account: consente di cancellare definitivamente l'account.
4. Lascia messaggi:
    - L'utente ha anche l'opzione di comunicare direttamente con i parrucchieri tramite dei messaggi.
5. Logout:
    - Per uscire dal sistema, l'utente può selezionare l'opzione Logout, che chiude la sessione.

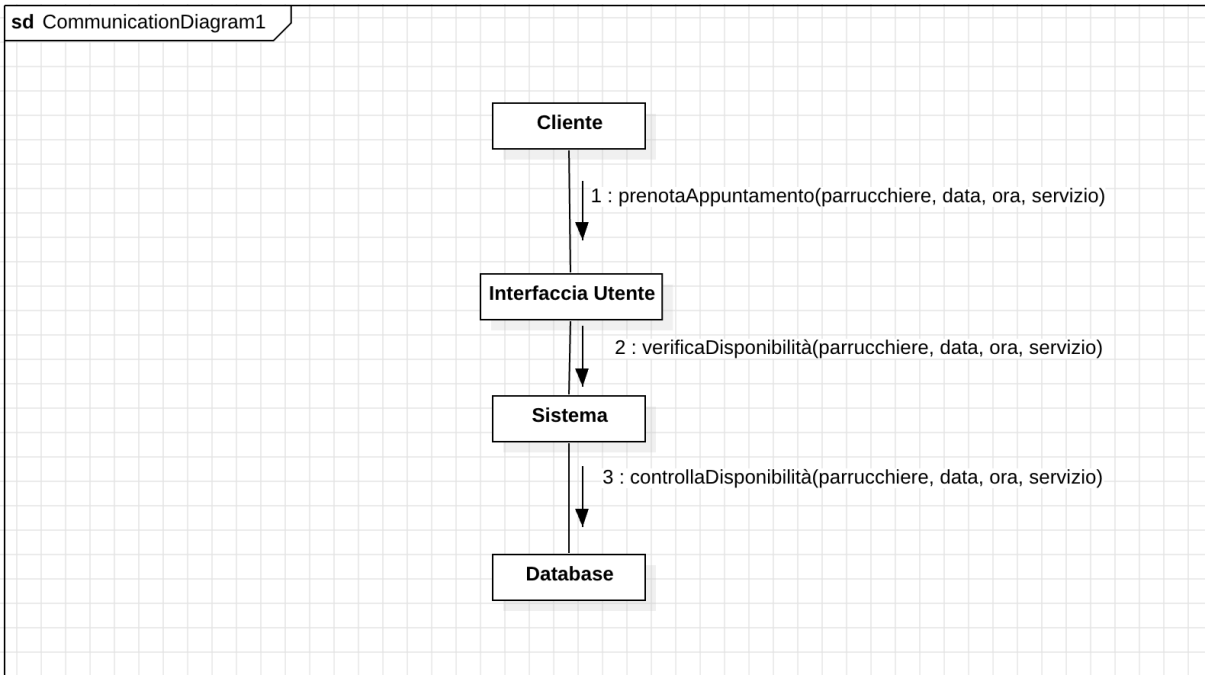


8.3 Diagramma di Comunicazione - Prenotazione Appuntamento

Il diagramma di comunicazione rappresenta le interazioni tra i vari componenti del sistema per il processo di prenotazione di un appuntamento. È stato progettato per evidenziare lo scambio di messaggi tra cliente, interfaccia utente, sistema e database, mostrando il flusso logico delle operazioni.

1. Cliente: Avvia il processo di prenotazione inviando una richiesta che include il parrucchiere, la data, l'orario e il servizio desiderato (prenotaAppuntamento).
2. Interfaccia Utente: Riceve la richiesta e la inoltra al sistema per la verifica della disponibilità (verificaDisponibilità).
3. Sistema: Interroga il database per controllare la disponibilità dei dati richiesti (controllaDisponibilità).
4. Database: Fornisce al sistema le informazioni necessarie per confermare o negare la prenotazione.

Questo diagramma è stato essenziale per definire il flusso di informazioni tra i componenti e garantire che ogni richiesta venga processata in modo corretto e coerente. Rende chiara la logica operativa e permette di individuare eventuali punti critici nel processo.

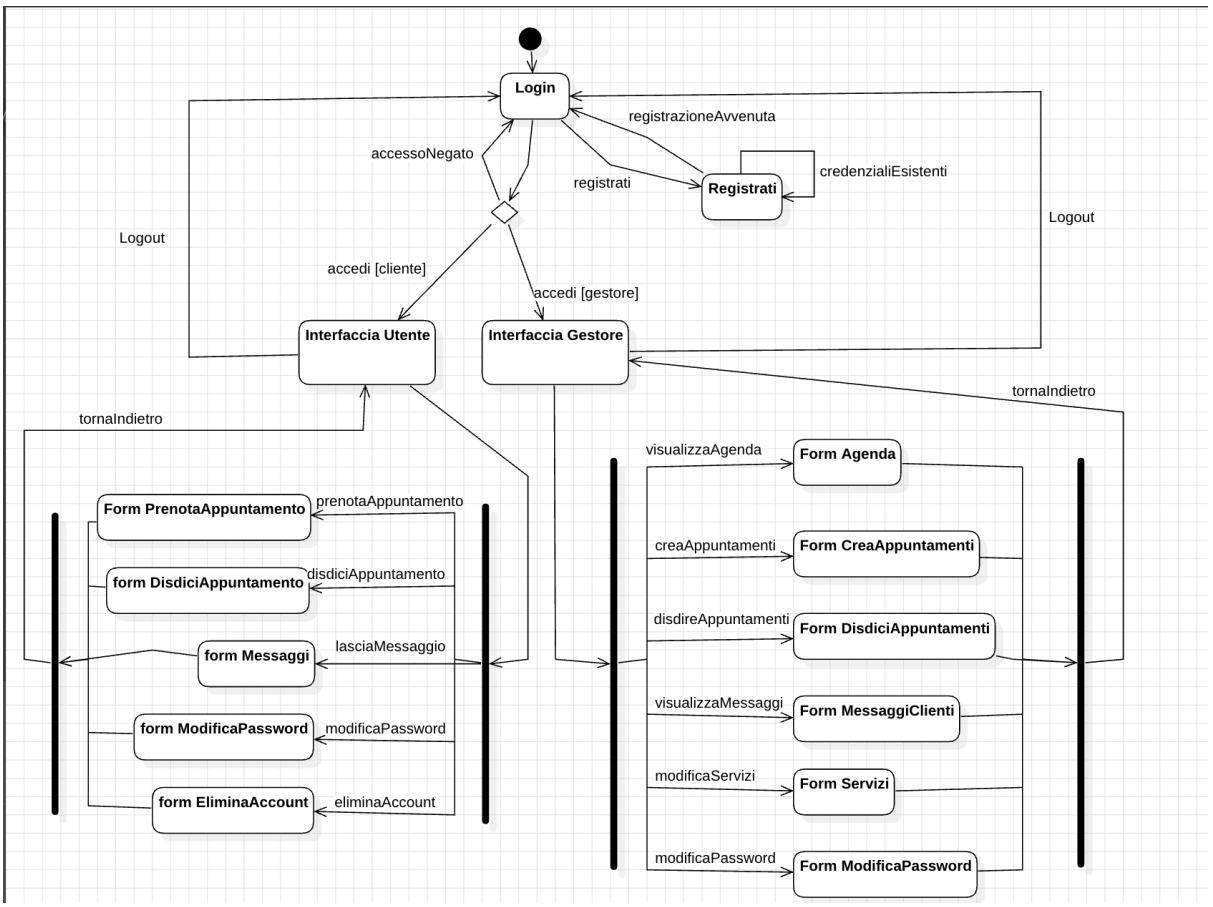


8.4 Diagramma di Stato - Gestione del Sistema

Il diagramma di stato rappresenta i diversi stati del sistema e le transizioni che avvengono in base alle azioni dell'utente. Evidenzia il flusso operativo principale dalla fase di login fino alla gestione delle funzionalità disponibili per clienti e gestori.

- Stato iniziale: L'utente inizia con il Login.
    - Accesso negato: Se le credenziali non sono valide, il sistema ritorna allo stato di login.
    - Registrazione: Se l'utente non è registrato, può accedere allo stato di Registrati per creare un nuovo account.
    - Accesso riuscito: L'utente viene indirizzato all'interfaccia appropriata:
        - Interfaccia Cliente: Permette di accedere a funzionalità come:
            - Prenotare o disdire appuntamenti.
            - Inviare messaggi al gestore.
            - Modificare la password o eliminare l'account.
        - Interfaccia Gestore: Consente di:
            - Visualizzare o creare appuntamenti.
            - Gestire i servizi offerti.
            - Rispondere ai messaggi dei clienti.
            - Modificare la propria password.
- Stati funzionali:
    - Form Prenota Appuntamento: Consente al cliente di selezionare data, orario e servizio.
    - Form Modifica Password: Per cambiare le credenziali d'accesso.
    - Form Agenda: Per il gestore, mostra gli appuntamenti settimanali.
- Stato finale: L'utente può eseguire il Logout, che chiude la sessione e riporta al login.

Questo diagramma aiuta a comprendere le transizioni chiave e i percorsi logici seguiti dagli utenti, sia clienti che gestori, durante l'utilizzo del sistema.



9. Software Architecture

Struttura MVC:

Model: Il model rappresenta i dati principali dell'applicazione e si occupa della loro gestione e manipolazione. Inoltre, interagisce con il controller per riflettere le modifiche nei dati e aggiornare la visualizzazione dell'interfaccia utente.

View: La view rappresenta l'insieme delle interfacce grafiche (GUI) che gestiscono l'interazione con l'utente. Attraverso queste interfacce, gli utenti possono effettuare prenotazioni, selezionare servizi e orari, e visualizzare informazioni. Gli input forniti dagli utenti vengono gestiti dal controller per essere elaborati.

Controller: Il controller funge da intermediario tra la View e il Model. Gestisce gli input degli utenti, come le prenotazioni o le modifiche di appuntamenti, validando la correttezza. Dopo l'elaborazione dei dati, il Controller aggiorna la View per riflettere i risultati delle operazioni, assicurando un feedback adeguato all'utente, sia esso una conferma o un avviso di errore.

Esempio di utilizzo nel progetto:
Un utente accede al sistema per prenotare un appuntamento con un parrucchiere. L'utente seleziona una data, un orario e i servizi desiderati attraverso l'interfaccia grafica (View). Quando l'utente conferma, l'input viene inviato al controller, che elabora la richiesta verificando la disponibilità attraverso il model. Il model, ad esempio, controlla se l'orario selezionato è disponibile e registra la prenotazione nei dati.
Se la prenotazione va a buon fine, il controller aggiorna il model e la view per riflettere i

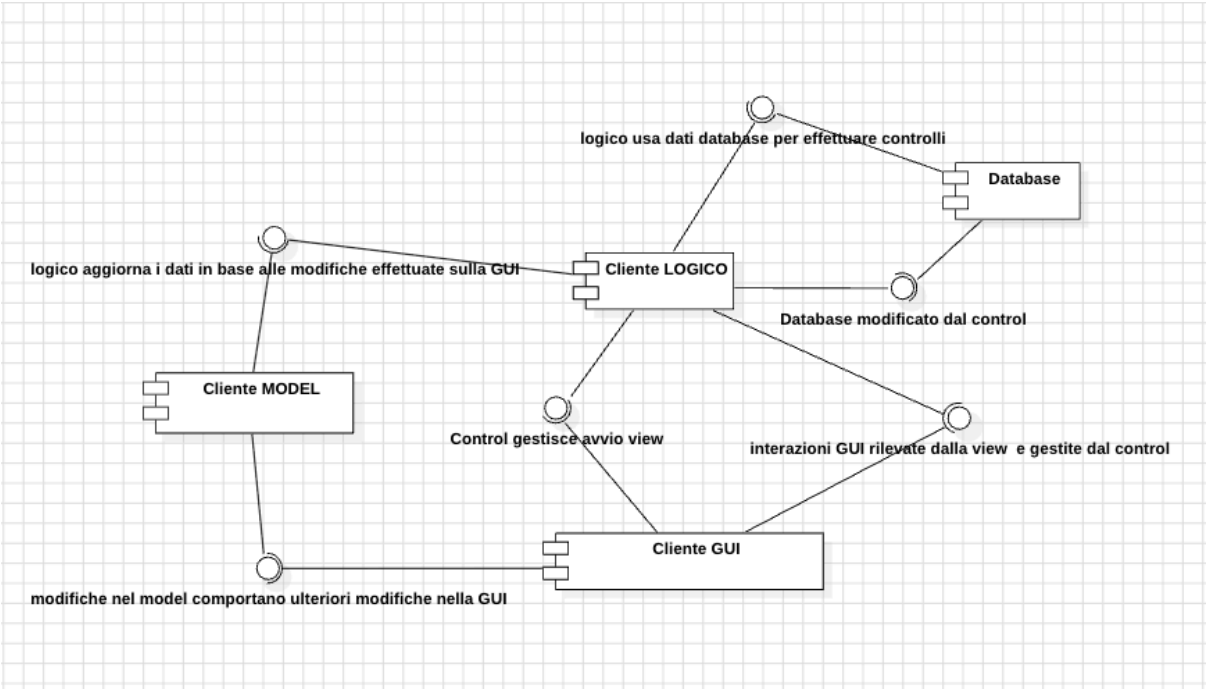
cambiamenti: la nuova prenotazione viene visualizzata nell'agenda del parrucchiere e l'orario prenotato risulta non più disponibile. Qualsiasi modifica nel model (ad esempio, un nuovo appuntamento o una cancellazione) si riflette immediatamente sulla view, assicurando un'interfaccia utente sempre aggiornata.

Diagramma dei Componenti del Sistema Cliente

Il diagramma rappresenta l'architettura modulare del sistema cliente, evidenziando le interazioni tra i componenti principali.

- Cliente GUI: Gestisce l'interfaccia utente, rilevando le interazioni e inoltrandole al componente di controllo (Control). Le modifiche effettuate dalla GUI influenzano direttamente il Cliente MODEL.
- Cliente LOGICO: Utilizza i dati del Database per eseguire controlli e aggiornare il sistema in base alle modifiche richieste dalla GUI.
- Cliente MODEL: Contiene i dati dell'applicazione. Eventuali modifiche al modello si riflettono sulla GUI.
- Control: Coordina l'avvio della vista, gestisce le interazioni tra GUI, LOGICO e MODEL, e modifica il Database quando necessario.
- Database: Archivio centrale per i dati; viene consultato e aggiornato dal componente logico e dal control.

La struttura garantisce una chiara separazione tra interfaccia, logica e archiviazione, rendendo il sistema modulare e facilmente estensibile.



Uso di Log4j:

Per garantire una gestione efficiente dei log, il nostro programma utilizza la libreria **Log4j**. Attraverso Log4j, abbiamo registrato informazioni importanti durante l'esecuzione

dell'applicazione, come eventi critici, errori, e dettagli utili per il debug. Ad esempio, quando un utente effettua una prenotazione, il sistema registra un log informativo che documenta l'operazione, mentre in caso di errori (come il tentativo di prenotare un orario non disponibile), viene generato un log di errore. Questo approccio ci ha permesso di monitorare l'applicazione in modo efficace e di identificare rapidamente eventuali problemi.

10. Software Design

Abbiamo utilizzato come design pattern il MVC e il Singleton:

- MVC come design pattern, ci riferiamo all'approccio di suddividere l'applicazione in tre componenti principali: Model, View e Controller. Questo pattern è principalmente orientato a migliorare l'organizzazione del codice e a separare le responsabilità all'interno del software. L'obiettivo è rendere i componenti più facili da sviluppare, testare e mantenere, isolando le varie funzionalità in blocchi logici distinti che interagiscono tra loro.

- Il Singleton è un design pattern che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a quell'istanza. Nel contesto delle classi DAO (Data Access Object), il Singleton viene utilizzato per gestire l'interazione con il database in modo centralizzato ed efficiente.

  Significato del Singleton per le classi DAO

1. Unica istanza condivisa: La classe DAO viene istanziata una sola volta durante il ciclo di vita dell'applicazione. Questo è utile per evitare la creazione di più connessioni al database, che potrebbero essere costose in termini di risorse.

2. Accesso globale: Tutte le parti del codice che necessitano di interagire con il database possono usare la stessa istanza DAO, garantendo un accesso uniforme.

3. Semplificazione della gestione delle risorse: Con un Singleton, è più facile gestire risorse come connessioni al database, pool di connessioni, o transazioni.

   Questo pattern è una soluzione standard per le applicazioni che richiedono interazioni frequenti con il database e garantisce una gestione più efficiente delle risorse.

Diagramma dei pacchetti

I diagrammi dei pacchetti sono diagrammi strutturali utilizzati per mostrare l'organizzazione e la disposizione di vari elementi del modello sotto forma di pacchetti. I diagrammi dei pacchetti sono comunemente utilizzati per fornire un'organizzazione visiva dell'architettura a strati.

1. Autenticazione: Necessario sia per il funzionamento sia di ‘InterfacciaCliente’ sia di ’InterfacciaGestore. Garantisce la gestione delle credenziali e l’accesso controllato
2. InterfacciaCliente e InterfacciaGestore: Questi due pacchetti rappresentano le interfacce utente, differenziate tra il cliente e il gestore del sistema.
3. Comunicazioni: Gestisce messaggi/notifiche/comunicazioni del sistema.
4. GestioneAppuntamenti: interagisce con il database per memorizzare e recuperare dati sugli appuntamenti.
5. GestioneServizi: interagisce con il database per le operazioni di gestione dei servizi offerti.
6. BaseDati: Contiene tutti i dati del sistema.

.

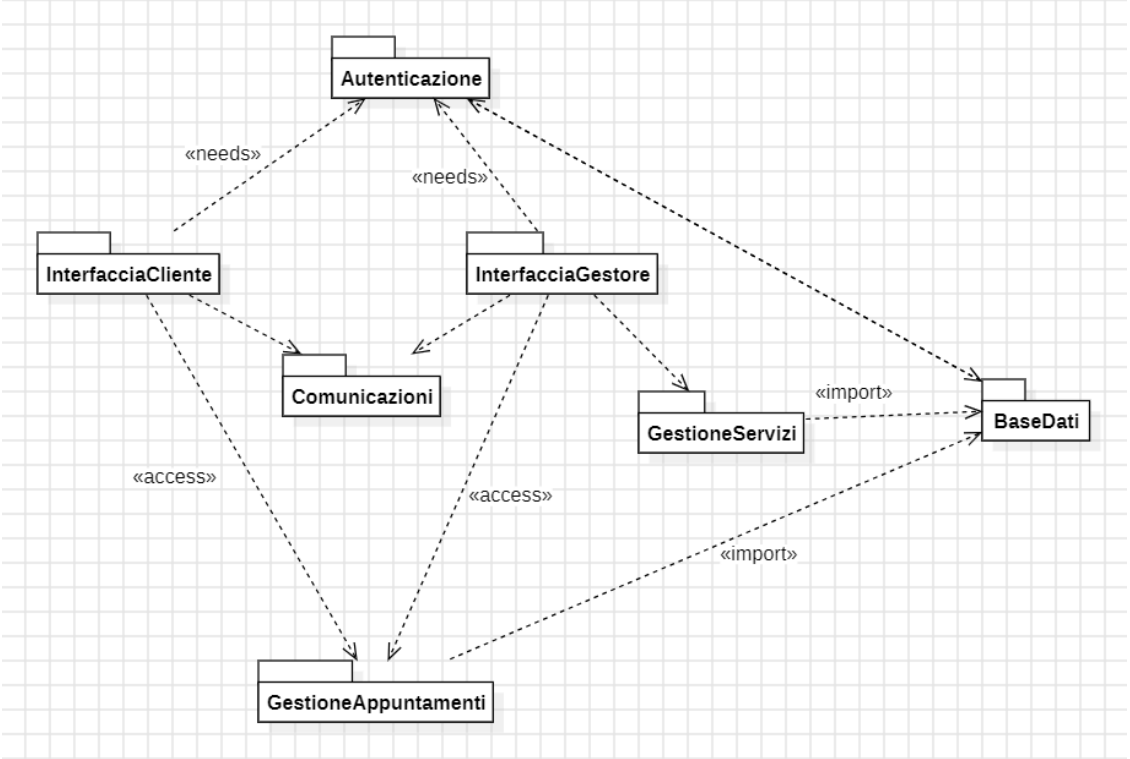


Diagramma delle Classi del Sistema

(Per motivi di spazio ed estetici non abbiamo riportato il diagramma in questo file, ma è possibile trovarlo sul nostro github)

Il diagramma delle classi rappresenta la struttura del sistema, suddivisa in tre pacchetti principali: View, Controller e Model, ognuno dei quali con ruoli e responsabilità ben definiti.

Pacchetto View

Questo pacchetto include tutte le classi che gestiscono l'interfaccia grafica del sistema. Ogni vista è progettata per gestire specifiche funzionalità:

- LoginView, RegisterView, ClientView, GestoreView, ecc.: Offrono finestre specifiche per l'interazione utente, come login, registrazione, gestione degli appuntamenti e visualizzazione dei messaggi.
- Ogni vista comunica con il corrispondente Controller per elaborare le azioni dell'utente.

Pacchetto Controller

Le classi di questo pacchetto gestiscono la logica applicativa, coordinando la comunicazione tra la View e il Model:

- CreateAppointmentController, ModifyServiceController, LoginController, ecc.: Controllano le operazioni principali, come creazione/modifica di appuntamenti, gestione dei servizi, login e registrazione.
- Ogni controller utilizza il modello per recuperare o aggiornare i dati e invia risposte alla view.

Pacchetto Model

Contiene le classi che rappresentano i dati e la logica di dominio del sistema:

- Appointment, User, Service, Message, ecc.: Modellano gli elementi centrali del sistema, con metodi per recuperare, aggiornare e salvare dati.
- DAO (Data Access Object): Le classi come AppointmentDAO, ServiceDAO gestiscono l'interazione con il database, assicurando una separazione tra logica di business e accesso ai dati.

Relazioni Principali

- Associazioni: Le View sono collegate ai Controller, che a loro volta interagiscono con il Model per garantire l’integrità dei dati.
- Ereditarietà: Alcune classi condividono comportamenti comuni grazie a relazioni di ereditarietà.
- Dipendenze: Le DAO comunicano direttamente con il database per effettuare operazioni CRUD (Create, Read, Update, Delete).

11. Software Testing

Durante lo sviluppo, il corretto funzionamento del codice è stato verificato costantemente attraverso test pratici sulle funzionalità del programma, utilizzando in particolare i test JUnit. La maggior parte dei test è stata concentrata sulle classi del controller, essendo il cuore della logica applicativa. Di fronte a eventuali errori, si è proceduto alla loro risoluzione, talvolta coinvolgendo altri membri del team per un supporto collaborativo. In alcuni casi, la correzione di bug ha richiesto modifiche più significative al programma, comportando un refactoring del codice per adattarlo alle nuove esigenze funzionali.

12. Software Maintenance

La **manutenzione del software** rappresenta una fase fondamentale del ciclo di vita dell'applicazione, ed è stata considerata con particolare attenzione durante lo sviluppo del nostro software.

Abbiamo strutturato il codice in modo modulare, seguendo il pattern MVC, per facilitare interventi futuri. Questa organizzazione consente di apportare modifiche mirate, ad esempio, aggiornare la logica (Controller) o migliorare l'interfaccia utente (View), senza impattare negativamente sugli altri componenti.

Durante la fase di manutenzione, abbiamo effettuato due tipi di manutenzione:

1. **Manutenzione correttiva:** correzione di bug emersi durante l'utilizzo. Grazie ai log generati tramite Log4j e ai test JUnit implementati, possiamo identificare e risolvere rapidamente eventuali malfunzionamenti.
2. **Manutenzione adattativa:** adeguamento del software a nuovi requisiti tecnologici o ambienti, come l'aggiornamento del database.

Grazie a questi interventi di manutenzione, il nostro programma rimane affidabile, scalabile e in grado di rispondere alle esigenze in evoluzione degli utenti e del contesto tecnologico.