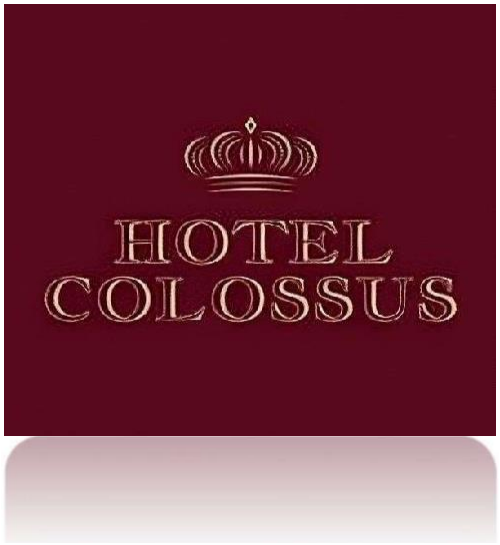
UNIVERSITÀ DEGLI STUDI
DI SALERNO



SYSTEM DESIGN DOCUMENT

TEAM COLOSSUS
Stefano Santoro – Giovanni Riccardi
-Renato Natale – Samuele Valiante

Data: 22/11/2025

Coordinatore del progetto:

Nome	Matricola
Stefano Santoro	0512120778

Partecipanti:

Nome	Matricola
Giovanni Riccardi	0512119392
Renato Natale	0512119641
Samuele Valiante	0512119125

Scritto da:	Giovanni Riccardi

Revision History

Data	Versione	Descrizione	Autore
25/11/2025	1.0	Prima versione SDD	Giovanni Riccardi
22/12/2025	1.1	Prima revisione SDD	Giovanni Riccardi

Sommario

INTRODUCTION

1.1 Purpose of the system

Il sistema software è progettato per la gestione alberghiera dell'hotel Colossus, che demanda un'architettura modulare e distribuita, pensata per supportare le diverse figure operative che lavorano all'interno dell'hotel. L'obiettivo principale dell'architettura è fornire una piattaforma unica, coerente, capace di gestire in modo integrato tutte le attività amministrative, operative e ristorative della struttura.

L'applicazione è organizzata in più sottosistemi cooperanti: gestione delle camere, gestione dei clienti e delle prenotazioni, gestione dello staff, contabilità, front desk, housekeeping e ristorazione. Questi sottosistemi comunicano tra loro attraverso delle interfacce che forniscono servizi esponendo operazioni ben definite, così da mantenere basso l'accoppiamento e permettere evoluzioni future senza impatti sull'intera applicazione.

L'architettura segue un modello client–server: lato client sono presenti le interfacce grafiche dedicate ai vari attori (front-desk, governante, manager, chef, maître), mentre lato server risiede la logica applicativa centrale che coordina i flussi dei dati, applica le regole di business ed effettua l'accesso al database. Il sistema software permette di mantenere separati l'aspetto visuale-grafico, la logica di controllo e la persistenza, garantendo manutenibilità e possibilità di estensioni future.

Nel complesso, la soluzione è pensata per essere portabile, sicura e intuitiva da usare. L'obiettivo che ci poniamo a raggiungere nel design è ottenere un sistema robusto, semplice da mantenere e facilmente estendibile, capace di adattarsi il più facilmente possibile a cambiamenti futuri qual ora fossero necessari. Infine, il documento serve come riferimento per lo sviluppo e l'implementazione del software, fornendo una guida chiara per i progettisti e futuri manutentori.

1.2 Design Goals

DESIGN GOAL DISCUSSION

I Design goals sono stati creati prendendo come base i requisiti non funzionali specificati durante la fase di requirements elicitation.
Dopo una attenta discussione abbiamo deciso di valorizzare con un focus prioritario l'efficienza operativa. Poiché, essendo un software per la gestione alberghiera, la velocità delle operazioni è un requisito fondamentale. Il sistema non deve essere la causa di rallentamenti durante i momenti più concitati e di affluenza , si deve quindi garantire velocità e affidabilità e le operazioni devono poter essere confermate nel modo più veloce possibile.

1.2.1 Portability

Il sistema deve essere progettato per minimizzare le differenze tra specifici sistemi operativi (come Windows, macOS, Linux), architetture di processore o configurazioni di rete. Il codice sorgente dovrebbe richiedere modifiche minime o nulle per essere compilato ed eseguito in ambienti eterogenei

1.2.2 Transparency

Il sistema deve essere distribuito con un architettura **client-server three tier** e deve essere sviluppato di modo che l'utente finale non percepisca questa proprietà. Il sistema distribuito deve essere inteso come un unico sistema logico.

1.2.3 Performance

- **Response time**

Le operazioni quotidiane: check-in, assegnazione camere, segnalazione pulizie devono essere confermate sotto il minuto per garantire un'esperienza utente immediata per il personale alberghiero. Questo è fondamentale per evitare colli di bottiglia durante i picchi di attività (ad esempio, l'orario di check-in). Qualsiasi ritardo percepibile oltre il limite specificato comporterebbe un aumento dei tempi di attesa per gli ospiti e una diminuzione della produttività del personale.

- **Throughput**

Il sistema deve supportare la concorrenza di più operazioni (es. front desk che registra un cliente mentre la governante aggiorna lo stato di una camera).Le operazioni sono viste come transazioni e come tali devono rispettare le proprietà ACID : Il sistema deve gestire efficacemente il carico di lavoro complessivo, mantenendo l'integrità dei dati e garantendo che le risorse condivise (come il database) abbiamo un failure rate più basso possibile

Throughput vs Response time

Per garantire che il sistema risponda entro il minuto deve essere assicurato che tra le varie operazioni concorrenti non vi siano problemi di deadlock, livelock e/o starvation. Tutte queste anomalie vengono affrontate durante l'implementazione che verterà sull'uso di strategie di programmazione concorrente.

1.2.4 Dependability

- **Robustness**

Deve gestire input errati e segnalarli all'utente (es. Quando il front desk inserisce dati errati gli viene notificato e data la possibilità di rimediare).

- **Reliability**

Le informazioni mostrate devono sempre corrispondere alla realtà operativa e a tutte le funzionalità che le riguardano.

- **Fault tolerance**

Il sistema deve poter fornire un certo grado di tolleranza ai malfunzionamenti, garantendo che se un sottosistema smetta di funzionare, non impatti significativamente verso gli altri.

Fault tolerance vs trasparency

Dato che Front-Desk e governante accedono alle stesse risorse, un malfunzionamento da parte di una componente condivisa da uno dei due lati deve essere percepito dall'altro come malfunzionamento generale nascondendogli la provenienza di esso, poichè non è di suo interesse.

- **Security**

Il sistema deve garantire accesso differenziato per ruolo (front desk, manager, governante, chef e maitre). Nessun dipendente deve poter accedere a dati non pertinenti e non riguardanti esso.
La protezione dei dati è un aspetto altrettanto importante: il sistema deve prevenire attacchi come SQL injection e crittografazione delle password nel database. (NFR9, NFR10)

1.2.7 Maintenance

- **Modifiability**

Deve essere garantita la facilità di aggiornare regole di business già esistenti.

Questo quando vi sono dei cambiamenti in una funzionalità o quando vi è un cambiamento delle politiche di operazione dei vari ruoli.

- **Readability**

Il codice dei vari moduli deve essere perfettamente chiaro, commentato e documentato per facilitare manutenzione e aggiornamenti da parte di sviluppatori futuri.

Performance vs. Maintenance

Verranno usati algoritmi e tecniche per ridurre ogni secondo di latenza. Il codice ottimizzato tenderà ad essere più denso. Saranno preferite le performance alla leggibilità del codice che potrà essere più veloce e meno pesante in termini di risorse.

1.2.8 End user criteria

- **Utility**

Il sistema deve supportare concretamente il lavoro quotidiano dell'impiegato fornendogli chiaramente uno stato dell'hotel riguardo le sue mansioni e facilitandogliele di modo da essere più efficiente nello svolgimento di quest'ultime.

- **Usability**

Poiché le attività alberghiere vengono svolte sotto pressione o in condizioni di lavoro dinamiche, l'interfaccia deve guidare gli operatori in modo naturale e senza ambiguità differenziate per ruolo. Il design del sistema punta quindi a fornire schermate chiare, suggerimenti testuali e messaggi di errore comprensibili, capaci di indicare all'utente la causa dell'errore e le azioni da intraprendere per correggerlo (NFR1, NFR2, NFR6).
La rappresentazione dello stato delle camere deve inoltre essere immediata e visiva, così da ridurre i tempi decisionali e aumentare l'efficienza operativa (NFR3).
Il manager deve avere anch'esso una interfaccia che gli permetta di avere una veduta generale dello stato attuale della struttura (dipendenti e conto economico) in modo intuitivo.
L'interfaccia di maître e chef permette di avere una panoramica continua e chiara sullo stato del servizio ristorativo; devono essere in grado di associare ad ogni tavolo una ordinazione.
La governante deve poter aggiornare lo stato di una camera con poche azioni in modo da comunicare velocemente e efficacemente lo stato della camera.

Performance vs Usability

Anche l'interfaccia si baserà sulla velocità di esecuzione delle operazioni essenziali (check-in, stato camera) al minimo di interazione. Si potrebbero sacrificare funzionalità di supporto come interfacce visivamente ricche, suggerimenti per l'operatore, se queste rallentano il sistema.

1.3 Definitions, acronyms, and abbreviations

Hotel Management System (HMS): Software progettato per gestire le operazioni quotidiane di un albergo, inclusi check-in, check-out, gestione camere, prenotazioni, servizi extra, personale e ristorazione.
Front Desk: Il cuore operativo dell'hotel dove i receptionist gestiscono le prenotazioni, i clienti e i servizi.
Housekeeping: Attività di pulizia, manutenzione e aggiornamento dello stato delle camere, svolta dalla governante.
Manager: Figura responsabile della supervisione delle operazioni, gestione del personale e del quadro economico della struttura.
Maître: Operatore responsabile della gestione degli ordini in sala, collegamento tra clienti e cucina.
Chef: Operatore responsabile della preparazione dei piatti e della gestione delle comande ricevute dalla sala.
Check-in: Procedura di registrazione del cliente all'arrivo in hotel e assegnazione della camera.
Check-out: Procedura di conclusione del soggiorno del cliente e pagamento dei servizi erogati dalla struttura.
Prenotazione: Registrazione di un soggiorno e/o di servizi aggiuntivi richiesti da un cliente.
Servizi Extra: Servizi aggiuntivi usufruiti dal cliente, come ristorazione in camera, spa o altri servizi a pagamento.
Ordine: Procedura mediante la quale il maître invia al servizio di cucina la comanda con i piatti da preparare, indicando quantità e tipologia. In questo momento la comanda diventa un effettivo ordine
Comanda: Registrazione sul sistema della richiesta del servizio ristorativo del cliente

Acronyms e abbreviations

- **HMS:** Hotel Management System
- **FR:** Functional Requirement (Requisito Funzionale)
- **NFR:** Non-Functional Requirement (Requisito Non Funzionale)

- **UI:** User Interface (Interfaccia Utente)
- **DB:** Database
- **SQL:** Structured Query Language

1.4 References

Il presente documento fa riferimento al Requirement Analysis Document (RAD), dal quale sono state tratte le specifiche sui requisiti funzionali, sui requisiti non funzionali e sui casi d'uso del sistema.

Current software application

Guardando i gestionali alberghieri attualmente sul mercato, appare chiaro che le soluzioni più usate e apprezzate integrano modularità, sicurezza e opzioni di deployment flessibili. La scelta di sviluppare un sistema distribuito in java mira a realizzare in maniera concreta le best pratice osservate nei sistemi gestionali alberghieri attualmente disponibili sul mercato.

HOASYS (GP Dati / Zucchetti)

- È un PMS (Property Management System) che può essere **in cloud** oppure in locale
- Ha modulo di ristorazione: nella loro documentazione è indicato che è possibile inviare "in tempo reale le comande raccolte ai vari reparti produttivi" tramite tablet wireless.
- Uso di database relazionale

Infinity Scrigno PMS (Zucchetti)

- Suite cloud per hotel indipendenti, catene, resort, con moduli per front-office, housekeeping, F&B, vendite, amministrazione, ecc.
- Usa un **database centralizzato** a cui accedono tutti i moduli applicativi.
- Architettura modulare e scalabile: moduli separati per i diversi reparti (front office, economato, ristorazione, etc.)

Hotelkit

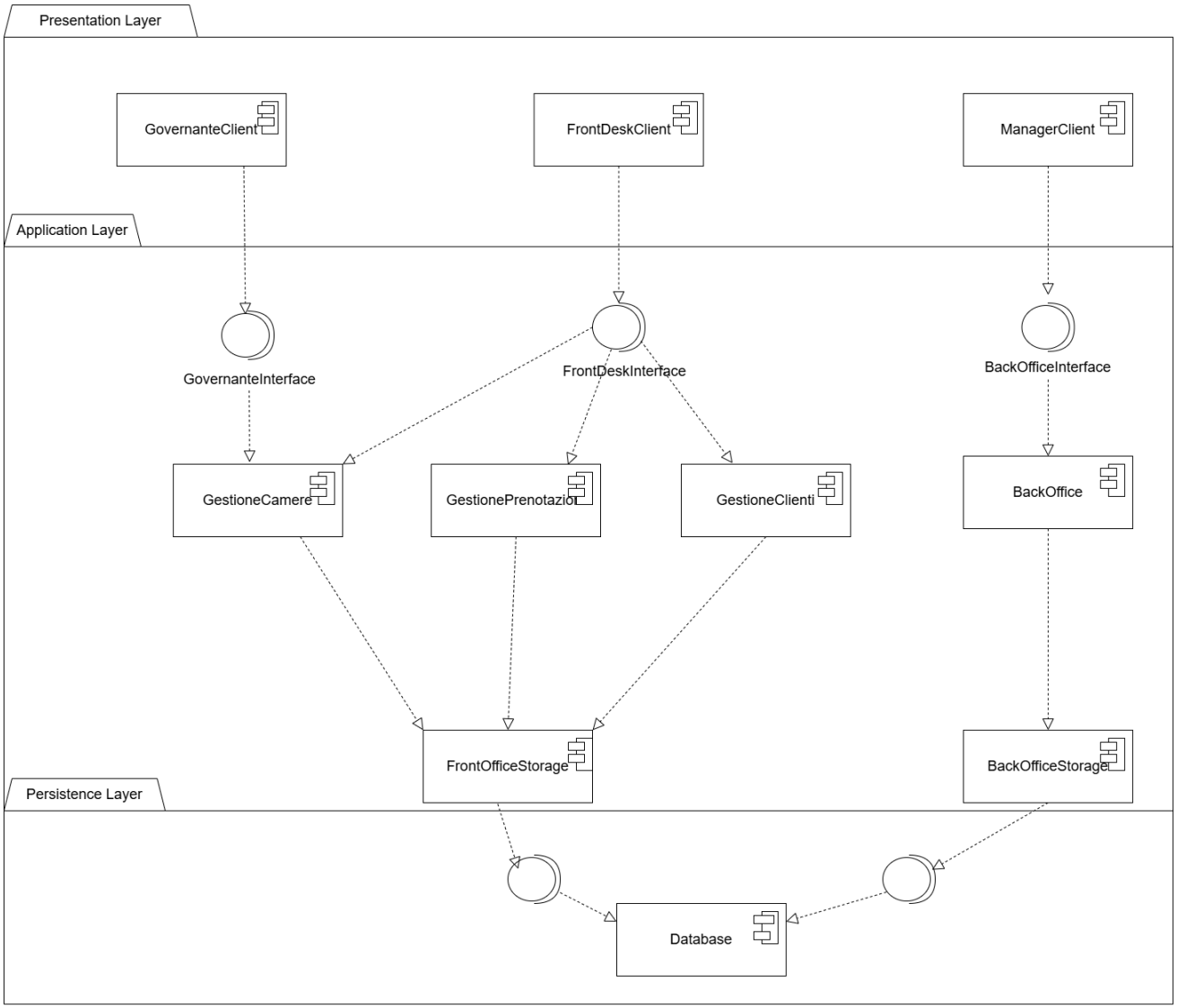
- È cloud-native, progettato per la cooperazione in tempo reale tra reparti,

con un modello distribuito per la comunicazione tra il front desk e il personale di servizio (governanti, manutentori).

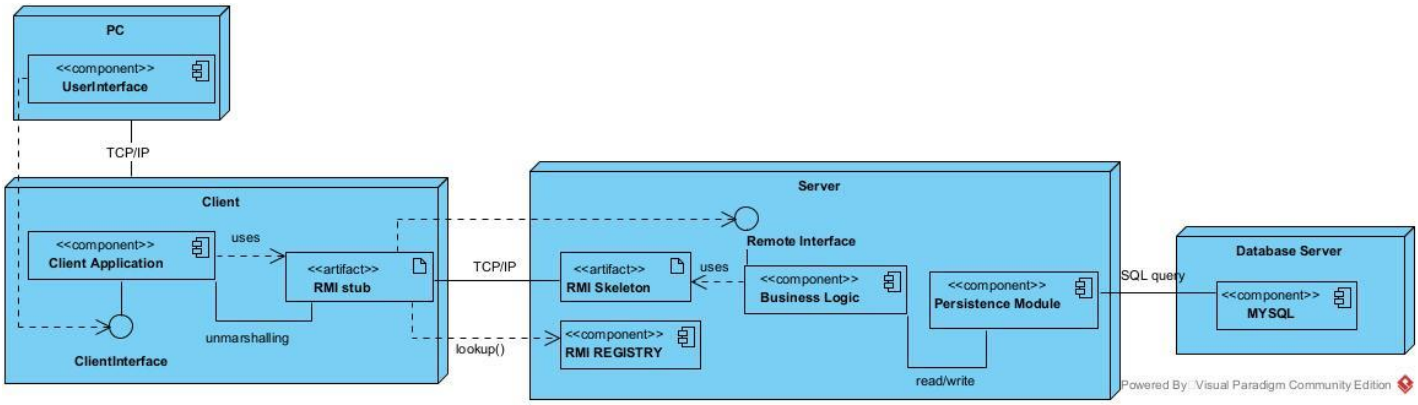
Proposed software architecture

Hardware software mapping

Component Diagram



Deployment Diagram



Persistent Data management

La gestione della persistenza viene implementata secondo un'architettura, in cui il Data Access Layer funge da intermediario tra la logica applicativa e il database relazionale.
DATABASE SCHEMA ALLEGATO AL DOCUMENTO

Abbiamo scelto di memorizzare le seguenti azioni tramite dei log file, questo per evitare di appesantire inefficientemente il database:

- Operatore Front Desk emette ricevuta
- Operatore Front Desk registra prenotazione
- Operatore Front Desk aggiunge servizio alla prenotazione
- Operatore Front Desk registra cliente
- Operatore Front Desk modifica prenotazione
- Manager inserisce un nuovo impiegato
- Manager rimuove un nuovo impiegato

Access Control and security

Attori \ Oggetti	Utente	ServizioAutenticazione	Impiegato	CatalogoImpiegati	Cliente	CatalogoClienti	Camera	CatalogoCamere	Prenotazione	CatalogoPrenotazioni	Servizio	RicevutaFiscale	Trattamento
Operatore front-desk		autenticazione cambioPassword			<<getter>> <<setter>>	registraCliente aggiornaDatiCliente cercaClienti ban / unBan getListaBannati			checkout aggiungiCliente aggiungiServizio rimuoviCliente rimuoviServizio	registraPrenotazione aggiornaDatiPrenotazione cercaPrenotazioni getStoricoPrenotazioni eliminaPrenotazione effettuaCheckOut	<<getter>> <<setter>>	<<getter>> <<setter>>	<<getter>> <<setter>>
Governante		autenticazione cambioPassword					<<getter>> setStato	setStatoOutOfOrder setStatoInPulizia getCamera					
Manager	setNewHashPassword	autenticazione cambioPassword	<<getter>> <<setter>>	registraImpiegato aggiornaDatiImpiegato eliminaImpiegato CercaImpiegati generaPasswordTemp									

Boundary conditions

Il nostro sistema software per gestire le problematiche relative alla manutenzione prevede una figura amministrativa, a cui viene dedicata una sezione per la gestione amministrativa delle problematiche.
L'amministratore dovrà effettuare le operazioni di avvio del software di shutdown tramite una interfaccia dedicata

UC1: Startup del sistema

Attore: Admin
Entry Condition: l'operatore si trova sulla sua interfaccia home
Flusso di eventi:

1. Clicca il pulsante "Start system"
2. Il sistema fa partire uno script che fa avviare l'RMI registry

Exit condition: Il sistema è avviato e gli altri operatori possono usare il sistema

UC2: Spegnimento del sistema

Attore: Admin
Entry Condition: l'admin deve stare nella sua schermata home
Flusso di eventi:

1. L'amministratore clicca il pulsante "Gestione del Software" e gli viene mostrata una schermata.
2. L'amministratore clicca il pulsante "shutdown System"

Exit condition: il sistema riceve il segnale di spegnimento del software e spegne il software

UC3: Configurazione del software

Attore: Admin
Entry Condition: l'admin deve stare nella sua schermata home
Flusso di eventi:

1. L'amministratore clicca il pulsante "Gestione del Software" e gli viene mostrata una schermata.
2. L'amministratore clicca il pulsante "shutdown System"

Exit condition: il sistema riceve il segnale di spegnimento del software e spegne il software

UC4: Avviso di manutenzione al sistema

Attore: Admin
Entry Condition: l'amministratore deve stare nella sua schermata home e deve essere effettuata una manutenzione nel sistema
Flusso di eventi:

1. L'amministratore clicca il pulsante "Comunicazione" e gli compare una piccola schermata.
2. L'amministratore scrive il messaggio (ad esempio: "in data 27 novembre alle ore 15:30 fino il 28 novembre alle 15:30 sarà effettuata una manutenzione").
3. successivamente inserisce nel sistema la data e l'ora
4. L'amministratore conferma l'operazione

Exit condition: il sistema ha memorizzato l'avviso

Ecco elencate le operazioni in caso di failure del sistema

-1) nel caso ci sia un problema nella ricezione di dati da parte del server il sistema inviterà al client di rieffettuare la richiesta
-2) l'amministratore in caso di manutenzione deve poter mandare un messaggio al client per indicare che sarà in corso una manutenzione fra un determinato arco di tempo
-3) il client nella sua interfaccia avrà una sezione dove riportare eventuali bug all'amministratore con una form di report.

Subsystem Servicies

GESTIONE CAMERE	
Nome servizio	**Servizi**
Modifica stato	setStatoOutOfOrder() setStatoInPulizia()
Recupera camera	getCamera()

GESTIONE PRENOTAZIONE	
Nome servizio	**Servizi**
Creazione prenotazione	registraPrenotazione(datiPren.)
Modifica prenotazione	aggiornaDatiPrenotazione(prenotazione) aggingiServizio(servizio) rimuoviServizio(servizio) aggingiCliente(cliente) rimuoviCliente(cliente)
Rimozione prenotazione	eliminaPrenotazione(prenotazione)
Recupero prenotazioni	getListaPrenotazioni(datiPren.)
Checkout prenotazione	effettuaCheckoutPrenotazione(datiPren.)

GESTIONE CLIENTI	
Nome servizio	**Servizi**
Registrazione cliente	registraCliente(datiCliente)

Modifica cliente	aggiornaDatiCliente(cliente)
Rimozione cliente	eliminaCliente(cliente)
Recupero clienti	getListaClienti(nome, cognome, nazionalità, dataNascita, Sesso)
Moderazione Clienti	ban(cliente) unBan(cliente) getListaBannati()

BACKOFFICE	
Nome servizio	**Servizi**
Registrazione impiegato	registraImpiegato(datiImpiegato)
Modifica impiegato	aggiornaDatiImpiegato(impiegato)
Rimozione impiegato	eliminaImpiegato (impiegato)
Recupero impiegato	getListaImpiegato (nome, cognome, sesso, ruolo)
Gestione accesso impiegati	generaPasswordTemporanea(impiegato)

AUTENTICAZIONE	
Nome servizio	**Servizi**
Autenticazione	autenticazione (username, password)
Recupero password	cambioPassword(newPassword, confirmPassword)