Università degli Studi di Trento
Facoltà di Informatica
Corso di Ingegneria del Software

FixMi
Documento di Architettura

Gruppo G43:
Giovanni Santini
Riginel Ungureanu
Valerio Asaro

Anno Accademico 2023/2024
Trento

CONTENTS

0.1 Scopo del documento

Il presente documento a titolo "Documento di Architettura" segue il documento "Specifica dei Requisiti". Il documento descrive il Diagramma delle Classi in "Unified Modeling Language" (UML), in modo continuativo al Diagramma delle Componenti e del Contesto, ai Requisiti Funzionali e Requisiti Non Funzionali, integrato a codice "Object Constraint Language" (OCL). Le classi definite dovranno essere implementate a livello di codice.

0.2 Informazioni del Documento

Campo	Valore
Titolo del Documento	Documento di Architettura
Titolo del Progetto	FixMi
Autori del Documento	Giovanni Santini Riginel Ungureanu Valerio Asaro
Amministratore Progetto	Riginel Ungureanu
Versione del documento	1.1

CHAPTER

ONE

DIAGRAMMA DELLE CLASSI

Nel presente capitolo vengono illustrate le classi previste nell'applicazione "FixMi". Tali classi sono rappresentate in "Unified Modeling Language" (UML) e sono caratterizzate da un nome, degli attributi specificando la visibilità ("+" = public, "-" = private) e dei metodi. Ogni metodo è caratterizzato da un valore di ritorno, un nome e degli argomenti. Inoltre alcune classi presentano un tipo al di sopra del nome per identificare proprietà particolari di un classe o aggiungere informazioni, questi sono:

- "Enumeration": L'oggetto rappresenta un valore che può essere in uno degli stati descritti all'interno dell'oggetto
- "Abstract": Una classe abstract non viene mai istanziata, viene perciò usata per effettuare templating e venire estesa da altre classi.
- "Singleton": Una classe che viene istanziata una ed una sola volta

Nei diagrammi che seguono viene rappresentato il rapporto tra diverse classi, utilizzando i simboli definiti sotto.

1.1 Definizione Simboli

Diagramma delle classi per i simboli

Definiamo i seguenti simboli riguardanti la relazione tra più classi:

- ISA: Vi è un rapporto di ereditarietà tra due classi
- Composizione: Stabilisce una relazione per cui una classe è parte di un'altra classe
- Aggregazione: Stabilisce una relazione per cui una classe possiede un'altra classe

1.2 Tipi di dato

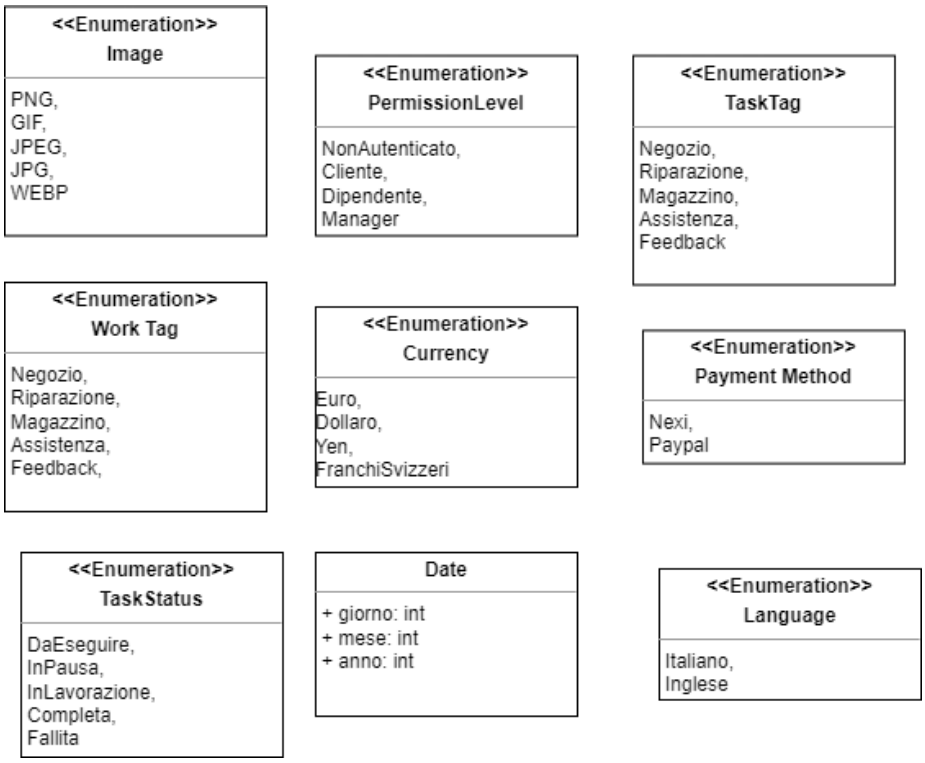

Diagramma delle classi per i tipi di dato

Allo scopo di definire dei tipi di dato chiari e comprensibili, sono state aggiunte le classi ed enumerazioni di cui sotto:

Image

L'Enumerazione Image rappresenta i diversi formati di immagine utilizzati all'interno dell'applicazione.

PermissionLevel

L'Enumerazione PermissionLevel rappresenta il livello di permesso di un certo utente

TaskTag

L'Enumerazione TaskTag rappresenta i vari tipi di TaskTag dell'applicazione

WorkTag

L'Enumerazione WorkTag rappresenta i vari tipi di WorkTag dell'applicazione

Currency

L'Enumerazione Currency rappresenta le valute supportate all'interno dell'applicazione

Payment Method

L'Enumerazione Payment Method rappresenta le piattaforme di pagamento utilizzabili all'interno dell'applicazione

TaskStatus

L'Enumerazione TaskStatus rappresenta lo stato di una determinata Task.

Date

La classe Date rappresenta una data

Language

L'Enumerazione Language rappresenta una lingua supportata dall'applicazione

1.3 Microservizio

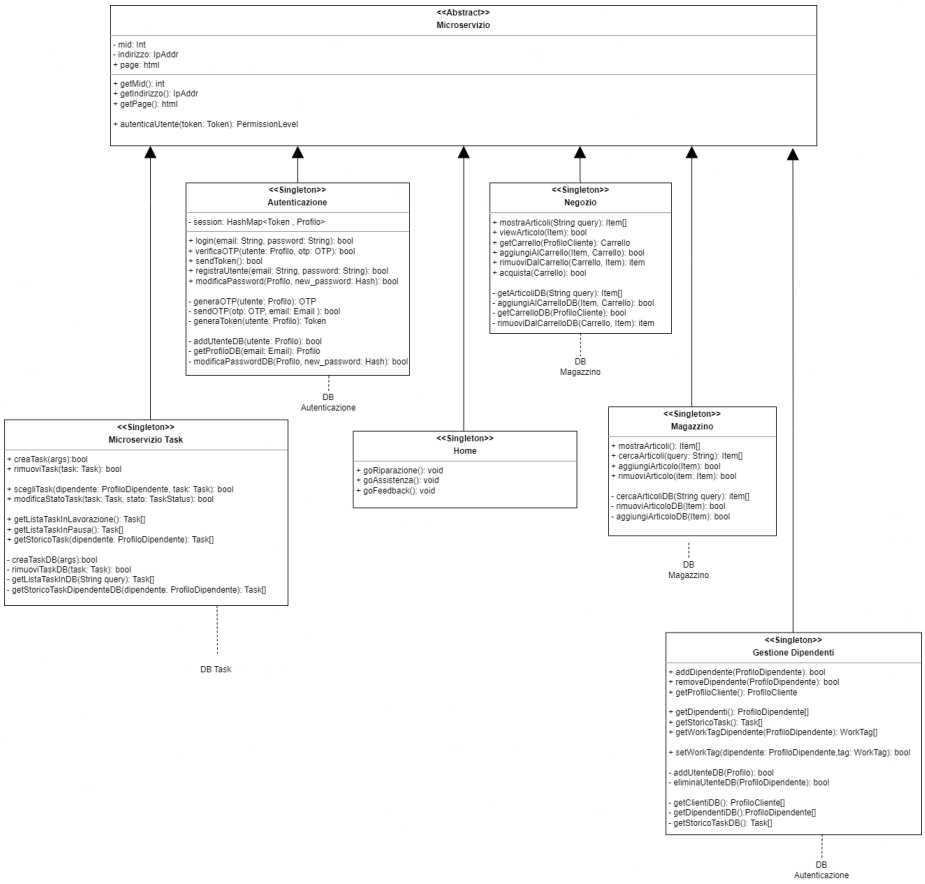

Diagramma delle classi per i Microservizi

Data la specifica del Back-End e dato il diagramma di contesto sono state individuate le classi e relativi metodi e attributi di cui sotto

Descrizione

La classe astratta Microservizio rappresenta un microservizio generico dell'applicazione. A tali fini sono stati identificati i seguenti attributi e metodi:

- private mid: Int
  - questo attributo è un codice identificativo univoco per il microservizio.
- private indirizzo: IpAddr

  - questo attributo corrisponde all'indirizzo IP e porta del microservizio.

- public page: html
  - questo attributo contiene la pagina principale del microservizio in html

- public autenticaUtente(token: Token): PermissionLevel
  - il metodo permette di richiedere al "microservizio autenticazione" il livello di permesso di un determinato utente, che ha fornito il suo token di sessione. ritorna il PermissionLevel dell'utente.

Classi figlie

Le classi figlie di "Microservizio" sono singleton, ciascuna rappresentante uno specifico microservizio dell'applicazione. Queste sono le seguenti:

- Microservizio Task
- Autenticazione
- Negozio
- Home
- Magazzino
- Gestione Dipendenti

1.4 Database

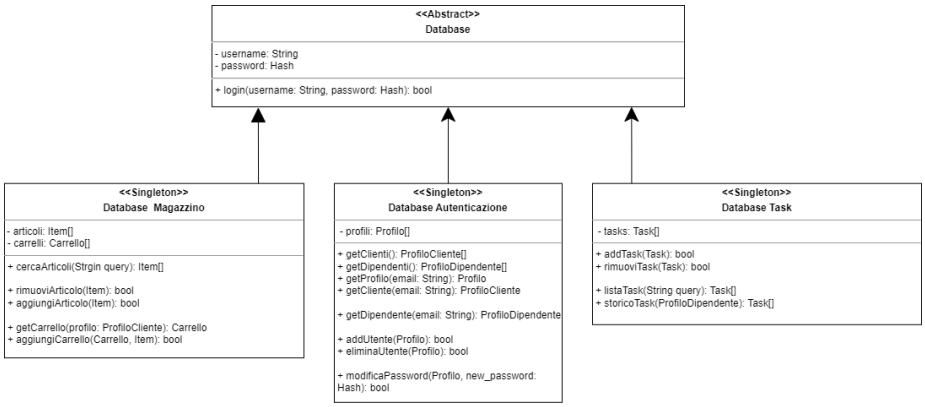

Diagramma delle classi per i Database

Data la specifica del Back-End e il diagramma di Contesto sono state individuate le classi e relativi metodi e attributi di cui sotto:

Descrizione

La classe astratta Database rappresenta un Database generico dell'applicazione. A questi fini sono stati identificati i seguenti attributi e metodi

- private username: String
  - questo attributo è l'username necessario per accedere al database
- private password: Hash
  - questo attributo è l'Hash della password necessaria per accedere al database
- public login(username: String, password: Hash): bool
  - questo metodo permette a un utente o a un microservizio di accedere al database per eseguire le proprie query. ritorna true se l'operazione è andata a buon fine, false altrimenti

Classi figlie

Le classi figlie di Database sono singleton, ciascuna rappresenta un specifico database dell'applicazione. Questi sono:

- Database Magazzino
- Database Autenticazione
- Database Task

1.5 Profilo

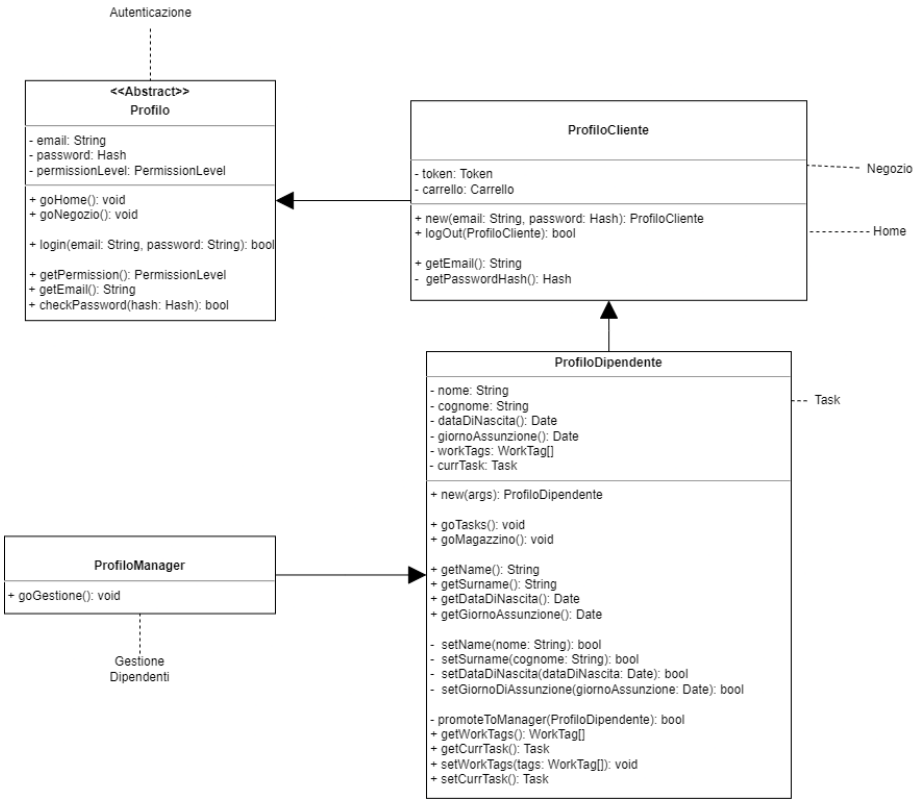

Diagramma delle classi per i profili

Data l'analisi del Contesto sezione 1 "Utenti e Sistemi Esterni" sono state individuate le classi e relativi metodi e attributi di cui sotto:

Descrizione

La classe astratta Profilo rappresenta il profilo di un utente del sistema che ha eseguito l'autenticazione. L'utente autenticato nel proprio profilo può utilizzare

le funzionalità del sistema attraverso i microservizi, e può visitare le varie pagine in base al proprio PermissionLevel. A questi fini sono stati identificati i seguenti metodi e attributi:

- private email: String
  - questo attributo è l'email appartenente all'utente
- private password: Hash
  - questo attributo è l'Hash della password dell'utente
- private permissionLevel: PermissionLevel
  - questo attributo è il livello di permesso dell'utente
- public goHome(): void
  - questo metodo permette all'utente di andare alla pagina Home
- public goNegozio(): void
  - questo metodo permette all'utente di andare alla pagina Negozio
- public checkPassword(hash: Hash): bool
  - questo attributo compara l'attributo password con l'hash fornito, ritornando true se combaciano, false altrimenti

ProfiloCliente

La classe ProfiloCliente, figlia della classe "Profilo" rappresenta il profilo di un cliente Sono stati identificati i seguenti metodi e attributi:

- private token: Token
  - questo attributo rappresenta il token di sessione di un cliente
- private carrello: Carrello
  - questo attributo rappresenta il carrello del cliente.
- public new(email: String, password: Hash): ProfiloCliente
  - questo metodo permette di creare un nuovo ProfiloCliente contenenti l'email e l'hash password forniti.

- public logOut(profilo: ProfiloCliente): bool
  - questo metodo permette di eseguire il logOut dall'applicazione, ritorna true se l'operazione è stata eseguita con successo, false altrimenti

ProfiloDipendente

La classe ProfiloDipendente, figlia della classe "ProfiloCliente" rappresenta il profilo di un dipendente dell'azienda. Sono stati identificati i seguenti metodi e attributi:

- private workTags: WorkTag[]
  - questo attributo rappresenta una lista di workTag del dipendente.
- private promoteToManager(profiloDipendente): bool
  - questo metodo permette di trasformare il profiloDipendente fornito in un profiloManager

ProfiloManager

La classe ProfiloManager, figlia della classe "ProfiloDipendente" rappresenta il profilo del manager dell'azienda.

1.6 Autenticazione

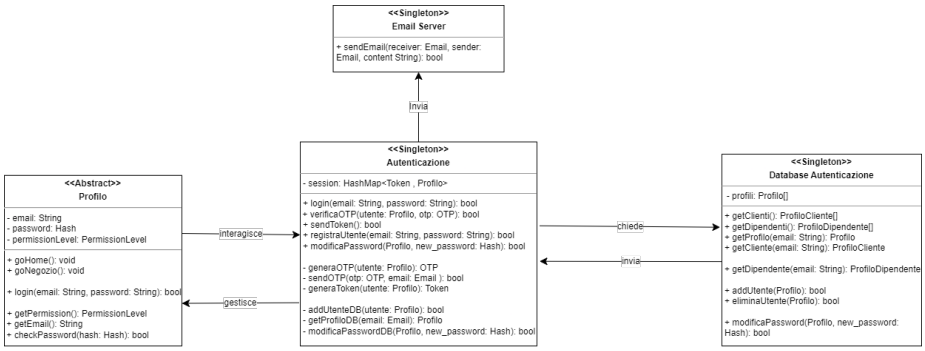

Diagramma delle classi per la componente "Autenticazione"

Date le seguenti componenti:

- "Autenticazione"
- "Login"
- "Registrazione"
- "Cambio Password"

seguendo ciò che è stato descritto negli RF "1" e "2" sono state individuate le classi e relativi metodi e attributi di cui sotto:

Descrizione

La classe "Autenticazione" è un singleton che rappresenta il microservizio avente lo stesso nome. Questa classe gestisce i dati di autenticazione dei profili quali e-mail, password e token. La classe contiene una HashMap che associa a ogni profilo un Token identificativo per la sessione. A questi fini, sono stati identificati i seguenti metodi:

- public login(email: String, password: String) : bool
  - questo metodo permette a un utente di eseguire il login, ritornando true se è avvenuto con successo e false altrimenti
  - all'interno del metodo, dopo aver eseguito il controllo dell'esistenza di un profilo con stessa email e password, viene generato un codice 2FA e inviato alla email rispettiva utilizzando i metodi "generaOTP()" e "sendOTP()", per poi aspettare che l'utente lo inserisca
  - alla fine del metodo e in caso di esito positivo, un token viene generato , inserito nella session e inviato all'utente utilizzando i metodi generaToken() e sendToken() rispettivamente
- public verificaOTP(utente: Profilo, otp: OTP): bool
  - questo metodo è necessario per verificare l'identità dell'utente attraverso il codice OTP, ritornando true se l'OTP inserito è giusto.
- public sendToken() :bool
  - questo metodo permette di inviare ad un utente il suo token
- public registraUtente(email: String, password: String): bool

  - questo metodo permette ad un nuovo utente di registrarsi, ritornando true se la registrazione è andata a buon fine
  - all'interno del metodo, dopo aver controllato che l'email fornita non appartenga a un profilo esistente, viene generato un codice 2FA e inviato alla email rispettiva utilizzando i metodi "generaToken()" e "sendToken()", per poi aspettare che l'utente lo inserisca

- public modificaPassword(Profilo, new_password: Hash) :bool
  - questo metodo permette ad un utente di modificare la propria password, fornendo l'hash della nuova password, e ritorna true se la modifica è andata buon fine, false altrimenti
  - all'interno del metodo viene generato un codice 2FA e inviato alla email rispettiva utilizzando i metodi "generaToken()" e "sendToken()", per poi aspettare che l'utente lo inserisca

- private generaOTP(utente: Profilo) : OTP
  - questo metodo genera un codice OTP per un profilo, ritornandolo

- private sendOTP(otp: OTP, email: Email): bool
  - questo metodo invia un codice OTP alla email fornita, ritornando true se l'operazione è andata a buon fine, false altrimenti

- private generaToken(utente: Profilo) : Token
  - questo metodo genera un token per un determinato utente, inserendolo nella session e ritornandolo

- private addUtenteDB(utente:Profilo): bool
  - questo metodo chiede al DB di aggiungere un nuovo profilo utente

- private getProfiloDB(email: Email) : Profilo
  - questo metodo chiede al DB il profilo avente la mail fornita. Se esiste viene ritornato il profilo, altrimenti null

- private modificaPasswordDB(profilo: Profilo, new_password: Hash) : bool
  - questo metodo richiede al DB la modifica della password di un dato profilo, e ritorna true se il processo è andato a buon fine, false altrimenti

Database Autenticazione

La classe Database Autenticazione è un singleton che rappresenta il database contenente i profili. Permette di gestire i profili, aggiungerli ed eliminarli. Sono stati individuati i seguenti metodi:

- public getClienti(): ProfiloCliente[]
  - questo metodo ritorna tutti i profili cliente immagazinati
- public getDipendenti(): ProfiloDipendente[]
  - questo metodo ritorna tutti i profili dipendente immagazinati
- public getProfilo(email: String): Profilo
  - questo metodo ritorna il profilo a cui corrisponde l'email fornita. Se non esiste ritorna null
- public getCliente(email: String): ProfiloCliente
  - questo metodo ritorna il profilo cliente a cui corrisponde l'email fornita. se non esiste ritorna null
- public getDipendente(email String): ProfiloDipendente
  - questo metodo ritorna il profilo dipendente a cui corrisponde l'email fornita. se non esiste ritorna null
- public addUtente(profilo: Profilo): bool
  - questo metodo inserisce il profilo fornito nel database. ritorna true se l'operazione è avvenuta con successo, false altrimenti
- public eliminaUtente(profilo: Profilo): bool
  - questo metodo elimina il profilo fornito dal database. ritorna true se l'operazione è avvenuta con successo, false altrimenti
- public modificaPassword(profilo: Profilo, new_password: Hash): bool
  - questo metodo modifica la password immagazzinata nel database del profilo fornito. ritorna true se l'operazione è avvenuta con successo, false altrimenti

Email Server

La classe Email Server è un singleton che rappresenta il server email utilizzato dall'azienda. permette di inviare email. Sono stati individuati i seguenti metodi:

- public sendEmail(receiver: Email, sender: Email,content: String) : bool
  - questo metodo invia una mail dall'email del mittente(sender) all'email del destinatario(receiver) con il contenuto(content)

Profilo

La classe astratta Profilo rappresenta il profilo di un determinato utente. Viene gestito dal microservizio Autenticazione e immagazzinato nel database Autenticazione. Sono stati individuati i seguenti metodi:

- public login(email: String, password: Hash) : bool
  - questo metodo permette a un utente di svolgere il login nel proprio profilo, fornendo l'email e la password. restituisce true se l'operazione è andata a buon fine, false altrimenti

1.7 Task

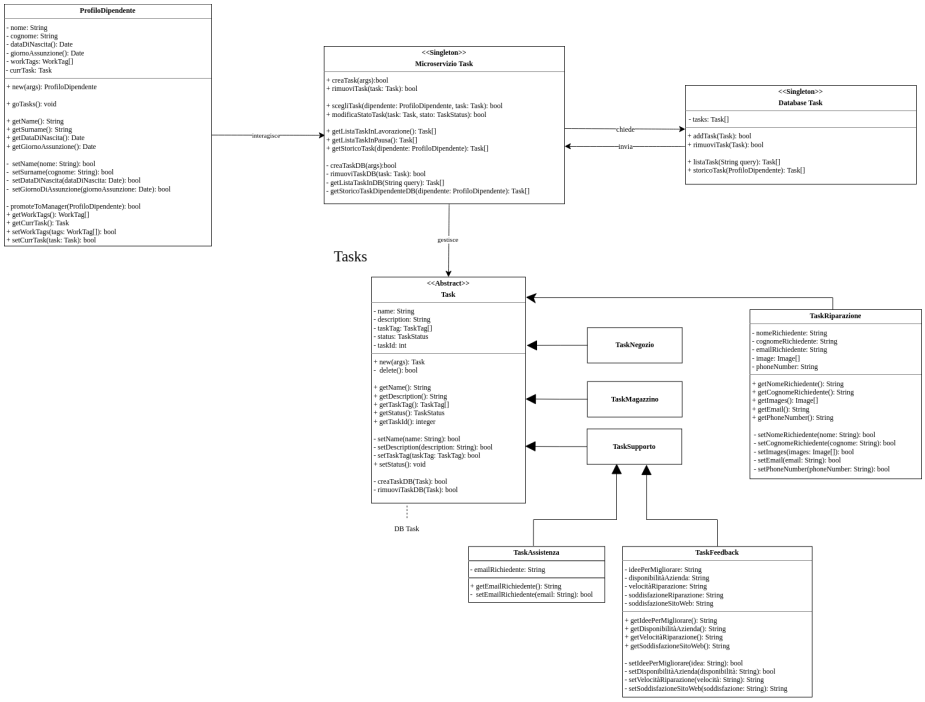

Diagramma delle classi per la componente "Task"

Data la componente "Task", seguendo ciò che è stato descritto nel Requisito Funzionale numero "9" e definita precedentemente nel rispettivo diagramma, sono state individuate le classi e relativi metodi e attributi di cui sotto:

Classe Task

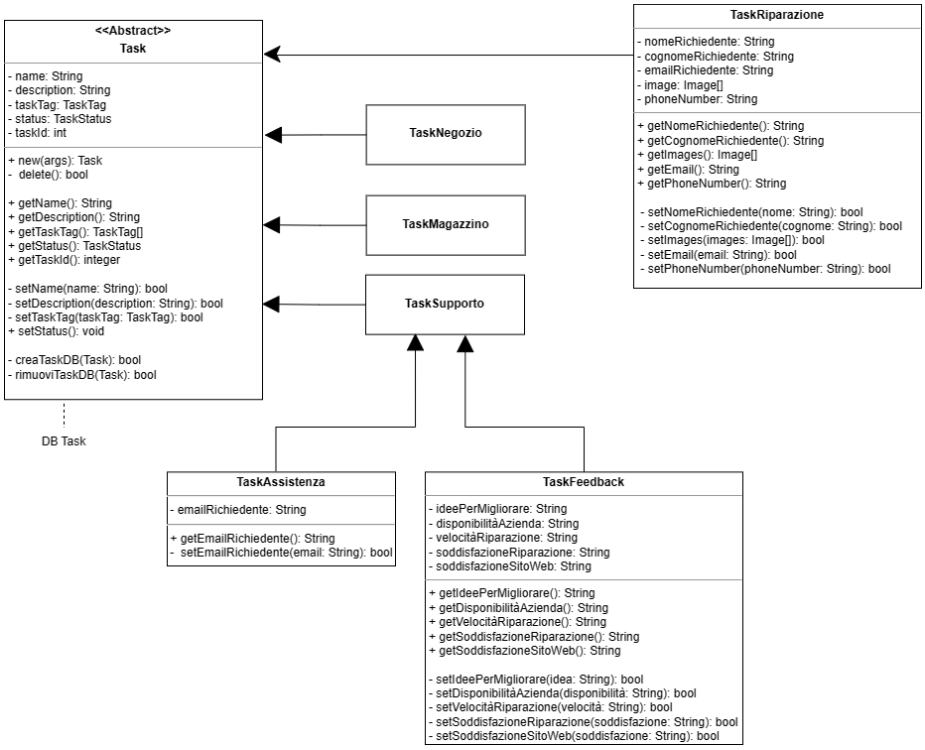

Diagramma delle classi per la componente "Task", 1

Descrizione

La classe "Task" definisce, come descritto dal Requisito Funzionale numero "9.1", l'oggetto "Task", da cui "TaskNegozio", "TaskMagazzino" e "TaskSupport" ereditano le proprietà. Le variabili di tipo "Task" vengono poi utilizzate dalla classe "Microservizio Task".

TaskRiparazione

Eredita dalla classe "Task", si specializza nella gestione delle Task per la Riparazione. Dato il Requisito Funzionale 9.1, sono stati individuati i seguenti metodi:

- private nomeRichiedente: String
  - Una stringa contenente un nome.

- private cognomeRichiedente: String
  - Una stringa contenente un cognome.
- private emailRichiedente: String
  - Una stringa contenente un indirizzo email.
- private image: Image[]
  - Contiene un gruppo di immagini.
- private phoneNumber: String
  - Una stringa contenente un numero di telefono.
- public getNomeRichiedente(): String
  - Ritorna una stringa contenente il nome del richiedente.
- public getCognomeRichiedente(): String
  - Ritorna una stringa contenente il cognome del richiedente.
- public getImages(): Image[]
  - Ritorna una stringa contenente le immagini della richiesta di riparazione.
- public getEmail(): String
  - Ritorna una stringa contenente l'email del richiedente.
- public getPhoneNumber(): String
  - Ritorna una stringa contenente il numero di telefono del richiedente.
- private setNomeRichiedente(nome: String): bool
  - Imposta il nome del richiedente, ritorna un valore booleano per indicare il successo dell'operazione.
- private setCognomeRichiedente(cognome: String): bool
  - Imposta il cognome del richiedente, ritorna un valore booleano per indicare il successo dell'operazione.

- private setImages(images: Image[]): bool
  - Imposta le immagini per la richiesta di riparazione, ritorna un valore booleano per indicare il successo dell'operazione.
- private setEmail(email: String): bool
  - Imposta l'email del richiedente, ritorna un valore booleano per indicare il successo dell'operazione.
- private setPhoneNumber(phoneNumber: String): bool
  - Imposta il numero di telefono del richiedente, ritorna un valore booleano per indicare il successo dell'operazione.

TaskAssistenza

Eredita dalla classe "Task", si specializza nella gestione delle Task per la Assistenza. Dato il Requisito Funzionale 9.1, sono stati individuati i seguenti metodi:

- private emailRichiedente: String
  - Una stringa contenente un indirizzo email.
- public getEmailRichiedente(): String
  - Ritorna una stringa contenente l'email del richiedente.
- private setEmailRichiedente(email: String): bool
  - Imposta l'indirizzo email del richiedente, ritorna un valore booleano per indicare il successo dell'operazione.

TaskFeedback

Eredita dalla classe "Task", si specializza nella gestione delle Task per il Feedback. Dato il Requisito Funzionale 9.1, sono stati individuati i seguenti metodi:

- private ideePerMigliorare: String
  - Una stringa contenente le idee per migliorare.
- private disponibilitàAzienda: String
  - Una stringa contenente la disponibilità dell'azienda.

- private velocitàRiparazione: String
  - Una stringa contenente la velocità della riparazione.
- private soddisfazioneRiparazione: String
  - Una stringa contenente la soddisfazione della riparazione.
- private soddisfazioneSitoWeb: String
  - Una stringa contenente la soddisfazione del sito Web.
- public getIdeePerMigliorare(): String
  - Ritorna una stringa contenente le idee per migliorare.
- public getDisponibilitàAzienda(): String
  - Ritorna una stringa contenente la disponibilità dell'azienda.
- public getVelocitàRiparazione(): String
  - Ritorna una stringa contenente la velocità dalla riparazione.
- public getSoddisfazioneRiparazione(): String
  - Ritorna una stringa contenente la soddisfazione della riparazione.
- public getEmail(): String
  - Ritorna una stringa contenente la soddisfazione del sito web.
- private setIdeePerMigliorare(idea: String): bool
  - Imposta una stringa contenente le idee per migliorare il sito, ritorna un valore booleano per indicare il successo dell'operazione.
- private setDisponibilitàAzienda(disponibilità: String): bool
  -
  - Imposta una stringa contenente le disponibilità dell'azienda, ritorna un valore booleano per indicare il successo dell'operazione.
- private setVelocitàRiparazione(velocità: String): bool
  - Imposta una stringa contenente la velocità della riparazione, ritorna un valore booleano per indicare il successo dell'operazione.

- private setSoddisfazioneRiparazione(soddisfazione: String): bool
  - Imposta una stringa contenente la soddisfazione della riparazione, ritorna un valore booleano per indicare il successo dell'operazione.
- private setSoddisfazioneSitoWeb(soddisfazione: String): bool
  - Imposta una stringa contenente la soddisfazione del sito Web, ritorna un valore booleano per indicare il successo dell'operazione.

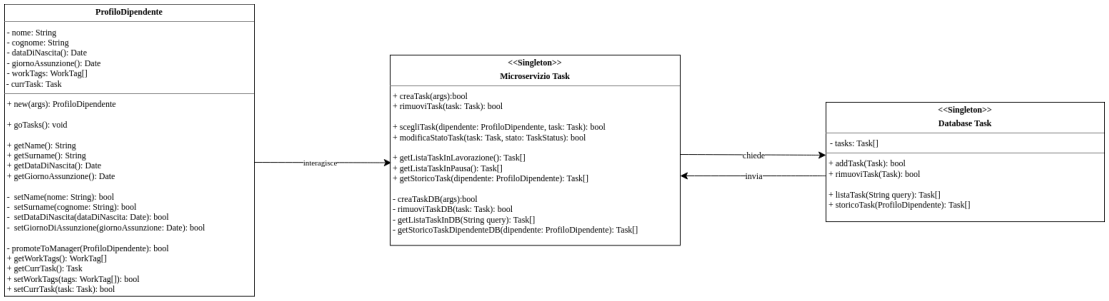

Diagramma delle classi per la componente "Task", 2

1.7.1 Microservizio Task

Descrizione

Microservizio Task

Si pone come tramite tra il dipendente ed il Database Task. Invia e riceve informazioni riguardanti le Task dal Database, permette al dipendente di creare una Task, rimuoverla o sceglierla (Cambiandone lo stato in "In Lavorazione"). Per tale scopo, sono stati individuati i seguenti metodi ed attributi:

- public creaTask(args):bool
  - Crea un nuovo oggetto di tipo "Task", ritorna un valore booleano per indicare il successo dell'operazione. Gli argomenti contengono i valori dei campi appartenenti alla classe "Task".
- public rimuoviTask(task: Task): bool
  - Rimuove un oggetto di tipo "Task", ritorna un valore booleano per indicare il successo dell'operazione. L'argomento è l'oggetto "Task" da eliminare.

- public scegliTask(dipendente: ProfiloDipendente, task: Task): bool
  - Sceglie un oggetto di tipo "Task" da assegnare ad un dipendente, fallisce nel caso in cui un dipendente stia ancora lavorando ad un altra Task.
- public modificaStatoTask(task: Task, stato: TaskStatus): bool
  - Modifica lo stato di un oggetto di tipo "Task", ritorna un valore booleano per indicare il successo dell'operazione.
- public getListaTaskInLavorazione(): Task[]
  - Ritorna una lista di tipo "Task" contenente oggetti "Task" che sono ancora "In Lavorazione".
- public getListaTaskInPausa(): Task[]
  - Ritorna una lista di tipo "Task" contenente oggetti "Task" che sono "In Pausa".
- public getStoricoTask(dipendente: ProfiloDipendente): Task[]
  - Ritorna una lista di tipo "Task" contenente oggetti "Task" completati da un dato dipendente.
- private creaTaskDB(args):bool
  - Salva i dati di un oggetto di tipo "Task" all'interno del Database, ritorna un valore booleano per indicare il successo dell'operazione.
- private rimuoviTaskDB(task: Task): bool
  - Rimuove i dati di un oggetto di tipo "Task" dal Database, ritorna un valore booleano per indicare il successo dell'operazione.
- private getListaTaskInDB(String query): Task[]
  - Ritorna una lista di tipo "Task" contenente il risultato di una richiesta effettuata sul Database.
- private getStoricoTaskDipendenteDB(dipendente: ProfiloDipendente): Task[]
  - Ritorna una lista di tipo "Task" contenente lo storico delle Task svolte da un dipendente, effettuando una richiesta al DataBase.

Database Task

La classe "Database Task" permette al Microservizio Task di interfacciarsi con il Database, fornisce metodi utili alla memorizzazione delle Task, alla loro gestione ed al loro recupero. I metodi individuati sono i seguenti:

- task: Task[]
  - L'insieme di oggetti "Task" memorizzato in maniera persistente.
- public addTask(Task): bool
  - Aggiunge un nuovo elemento di tipo "Task" al Database, ritorna un valore booleano per indicare il successo dell'operazione.
- public rimuoviTask(Task): bool
  - Rimuove un elemento esistente di tipo "Task" dal Database, ritorna un valore booleano per indicare il successo dell'operazione.
- public listaTask(String query): Task[]
  - Ritorna una lista di tipo "Task", risultato della ricerca fatta all'interno del Database tramite la stringa "query".
- public storicoTask(ProfiloDipendente): Task[]
  - Ritorna una lista di tipo "Task" contenente tutte le task completate dal profilo dipendente richiesto.

Profilo Dipendente

Il dipendente può accedere alla pagina "Tasks"(in cui le può visualizzare, gestire, creare ed eliminare) attraverso il seguente metodo:

- public goTasks(): void
  - Porta il dipendente nella pagina per gestire le Task.

1.8 Home

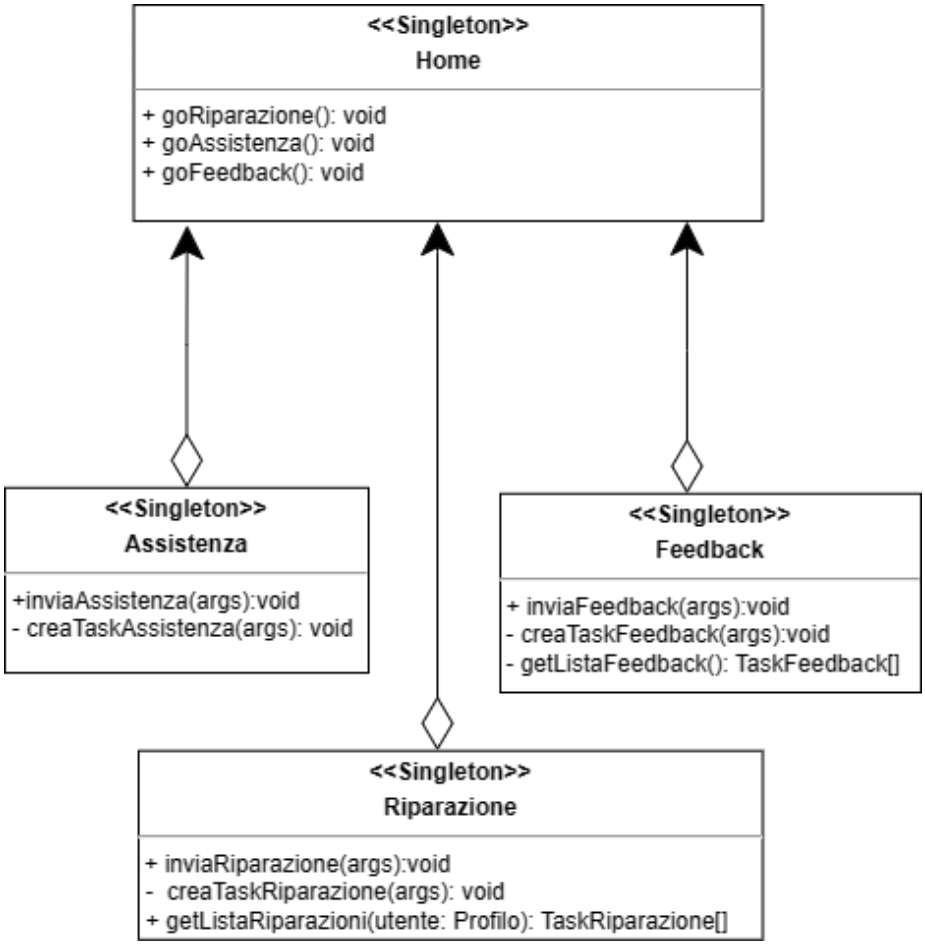

Diagramma delle classi per "Home"

Svolge la funzione di accogliere i visitatori nel sito "FixMi" e di permettere loro di accedere ad alcuni dei servizi principali quali:

- Richiedere assistenza al personale.
- Richiedere una riparazione del proprio prodotto ai tecnici.
- Inviare feedback riguardo alla qualità dei servizi offerti dallo staff e dal sito web.

Home

Descrizione

Rappresenta la prima pagina mostrata all'utente o chi, per la prima volta, visita il sito "FixMi". Da qui, è possibile raggiungere tutti gli altri servizi attraverso i metodi descritti sotto:

- public goRiparazione(): void
  - Manda il browser alla pagina "Riparazione".
- public goAssistenza(): void
  - Manda il browser alla pagina "Assistenza".
- public goFeedback(): void
  - Manda il browser alla pagina "Feedback".

Riparazione

Descrizione

Rappresenta la pagina in cui è possibile richiedere una riparazione dai tecnici "FixMi". I metodi individuati sono:

- public inviaRiparazione(args): void
  - Manda la richiesta di riparazione compilata dall'utente al sistema.
- private creaTaskRiparazione(args): void
  - Crea una nuova Task di tipo "Riparazione", contiene le informazioni specifiche alla riparazione stessa passategli per argomenti.
- public getListaRiparazioni(utente: Profilo): TaskRiparazione[]
  - Ritorna la lista delle richieste di riparazioni già effettuate da un dato utente passato come argomento al metodo.

Assistenza

Descrizione

Rappresenta la pagina in cui è possibile richiedere una assistenza dai tecnici "FixMi". I metodi sono:

- public inviaRiparazione(args): void
  - Manda la richiesta di assistenza compilata dall'utente al sistema.
- private creaTaskAssistenza(args): void
  - Crea una nuova Task di tipo "Assistenza" contenente le informazioni specifiche alla assistenza passando le informazioni necessarie per argomento.

Feedback

Descrizione

Rappresenta la pagina in cui è possibile mandare un "feedback" riguardo al servizio "FixMi". I metodi individuati che verranno utilizzati sono:

- public inviaFeedback(): void
  - Manda il feedback compilato dall'utente al sistema.
- private creaTaskFeedback(args): void
  - Crea una nuova Task di tipo "Feedback", contiene le informazioni specifiche al feedback stesso tramite argomenti passati al metodo.
- private getListaFeeback(): TaskFeedback[]
  - Ritorna la lista delle Task feedback presenti nel sistema.

1.9 Carrello

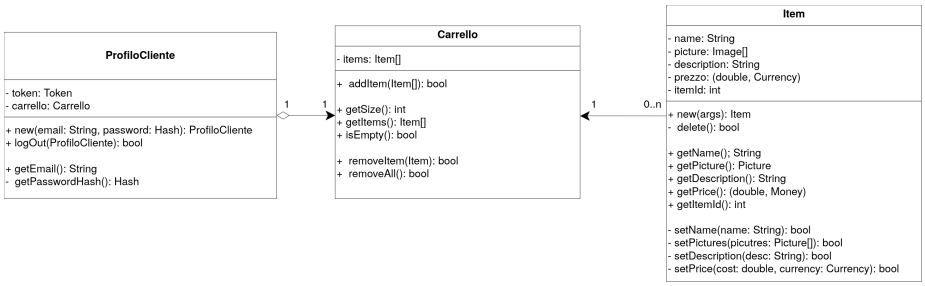

Diagramma delle classi per la componente "Carrello", mostrando le classi "Carrello", "Item", "Profilo Cliente" e "Database magazzino".

Descrizione

Data la componente "Carrello", definita nei documenti precedenti, in riferimento ai Requisiti Funzionali numero "5.1" e "5.2", sono state individuate le classi "Carrello" e "Item". La classe Carrello contiene una lista di "Items" ed offre una serie di funzionalità come l'elencazione, l'aggiunta e la rimozione di Items dalla lista. Inoltre, ogni "Profilo Cliente", definito in precedenza, è composto anche da uno ed uno solo Carrello. A questi fini, sono stati individuati i seguenti metodi per la classe Carrello:

- addItem(Item[]): bool
  - Questo metodo permette di aggiungere uno o più Items al Database. Ritorna true se l'operazione termina con successo senza error, altrimenti ritorna flase.
- getSize(): int
  - Questo metodo ritorna il numero di Items presenti nel carrello.
- getItems(): Item[]
  - Questo metodo ritorna la lista di Item presente nel carrello.
- isEmpty(): bool
  - Questo metodo ritorna true se non ci sono elementi nel carrello, altrimenti ritorna false.
- removeItem(Item): bool

  - Dato un Item come parametro, questo metodo rimuove lo stesso dalla lista degli Items nel carrello, ritornando il risultato dell'operazione.

- removeAll(): bool
  - Questo metodo rimuove tutti gli elementi dal carrello.

Item

Un Item è un'oggetto presente nel magazzino, che può essere acquistato. Sono stati individuati i seguenti metodi e attributi:

- name: String
  - Il nome dell'Item.
- picture: Image[]
  - Una lista di immagini associate all'Item.
- description: String
  - Una descrizione dell'Item
- prezzo: (double, Currency)
  - Il prezzo, salvato come una coppia di un numero reale (double) e una Currency
- itemId: int
  - Un numero identificativo ed univoco dell'Item
- new(args): Item
  - Questo metodo svolge la funzione di costruttore dell'Item, gli attributi vengono passati come argomenti alla funzione stessa. Ritorna un nuovo oggetto Item con i alori passati.
- delete(): bool
  - Questo metodo permette di eliminare l'istanza dell'oggetto Item.

Segue un'elencazione di metodi getter e setter degli attributi specificati sopra:

- getName(): String
- getPicture(): Picture
- getDescription(): String
- getPrice(): (double, Money)
- getItemId(): int
- setName(name: String): bool
- setPictures(picutres: Picture[]): bool
- setDescription(desc: String): bool
- setPrice(cost: double, currency: Currency): bool

ProfiloCliente

Ogni profilo cliente è composto da uno e uno solo carrello, inizialmente inizializzato con una lista vuota di Items. Il Cliente potrà accedere ed interagire con il proprio carrello nella pagina negozio speficata successivamente.

1.10 Negozio

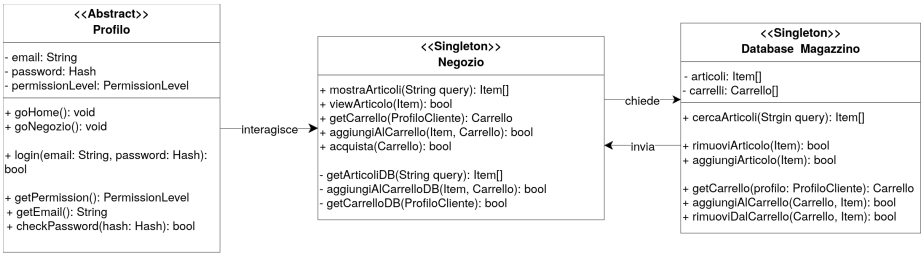

Diagramma delle classi per la componente "Negozio" in relazione alle classi "Database Magazzino" e "Profilo"

Descrizione

Data la componente "Negozio", definita nel diagramma delle componenti, è stata individuata la classe "Negozio". Tale classe si pone come tramite tra l'utente e il Database Magazzino, dando la possibilità di visionare gli articoli

presenti nel magazzino e, qualora l'utente fosse autenticato, di aggiungere / rimuovere articoli dal carrello. A tali fini, in riferimento alla componente "Negozio" citata sopra, vengono fortini i seguenti metodi pubblici:

- mostraArticoli(String query): Item[]
  - Questo metodo ritorna una lista di Item data una query al DB.
- viewArticolo(Item): bool
  - Questo metodo permette di visualizzare le informazioni relative ad un'articolo nella pagina del negozio. Ritorna true se l'operazione avviene con successo, altrimenti false.
- getCarrello(ProfiloCliente): Carrello
  - Dato un ProfiloCliente, questo metodo ritorna il carrello associato al profilo.
- aggiungiAlCarrello(Item, Carrello): bool
  - Questo metodo permette di aggiungere un item al carrello, dati questi elementi, interfacciandosi al databse. Ritorna true se il procedimento è andato a buon fine, false altrimenti.
- acquista(Carrello): bool
  - Questo metodo permette di acquistare gli item inseriti nel carrello. Ritorna true se il procedimento è andato a buon fine, false altrimenti.

Questi metodi si interfacciano alla classe Database Magazzino utilizzando i seguenti metodi privati:

- getArticoliDB(String query): Item[]
- aggiungiAlCarrelloDB(Item, Carrello): bool
- getCarrelloDB(ProfiloCliente): bool

Database Magazzino

La classe Database Magazzino ha il compito di interfacciarsi direttamente con il Database per salvare in modo persistente gli articoli e i carrelli. la classe presenta dei metodi pubblici per interagire con la base di dati come setters e getters.

In particolare sono stati individuati i seguenti metodi:

- cercaArticoli(Strgin query): Item[]
  - Questo metodo permette di interrogare il database tramite una query, ritornando il risultato dell'interrogazione.
- rimuoviArticolo(Item): bool
  - Questo metodo permette di rimuovere un'articolo dal database. Ritorna true solo se l'operazione è terminata con successo, false altrimenti.
- aggiungiArticolo(Item): bool
  - Questo metodo permette di aggiungere un'articolo ad database. Ritorna true solo se l'operazione è terminata con successo, altrimenti false.
- getCarrello(profilo: ProfiloCliente): Carrello
  - Questo metodo ritorna il carrello associato ad un ProfiloCliente.
- aggiungiCarrello(Carrello, Item): bool
  - Questo metodo permette di aggiungere un item al carrello. Richiede come parametri questi stessi e ritorna true se l'operazione è avvenuta con successo, altrimenti false.

Profilo

Ogni utente è in grado di accedere al Negozio, dunque è stato individuato il metodo "goNegozio()" che permetterà all'utente anche non autenticato di accedere alla pagina Negozio.

Il seguente diagramma riassume quanto detto precedentemente riguardo le classi Negozio, Database Magazzino, Profilo Cliente, Carrello, Item.

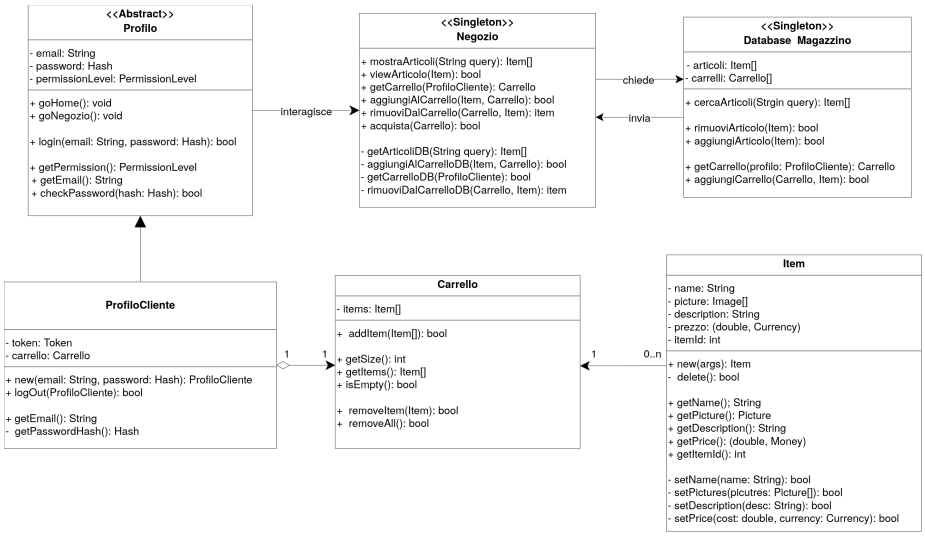

Diagramma delle classi per la componente "Negozio" in relazione alle classi "Database Magazzino" e "Profilo Cliente"

1.11 Magazzino

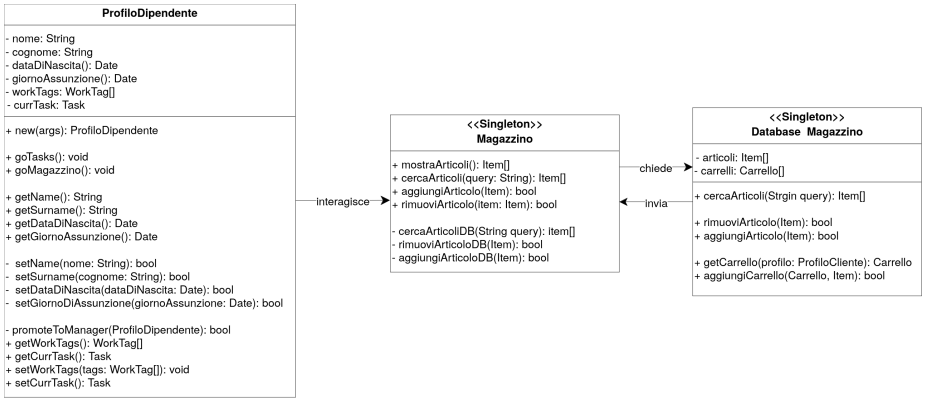

Diagramma delle classi per la componente "Magazzino", mostrando le classi "Magazzino", "ProfiloDipendente" e "Database Magazzino".

Descrizione

Data la componente Magazzino, secondo quanto definito in Requisito Funzionale numero "10", viene individuata la classe Magazzino. Tale classe è composta da una pagina Magazzino e permette al dipendente di interagire con il Database

Magazzino, aggiungendo e rimuovendo Articoli dall stesso e dando la possibilità di cercare tali articoli attraverso interrogazioni al database.

Vengono definiti i seguenti metodi:

- mostraArticoli(): Item[]
  - Questo metodo ritorna tutti gli articoli presenti nel Database Magazzino.
- cercaArticoli(query: String): Item[]
  - Questo metodo permette di interrogare il Database Magazzino passando come argomento una stringa "query" che contiene la richiesta.
- aggiungiArticolo(Item): bool
  - Questo metodo permette di aggiungere un oggetto Item al Database Magazzino.
- rimuoviArticolo(Item): bool
  - Questo metodo permette al dipendente di rimuovere un oggetto Item dal Database Magazzino.

La classe Magazzino interagisce con il Database Magazzino attraverso i metodi:

- cercaArticoliDB(String query): item[]
- rimuoviArticoloDB(Item): bool
- aggiungiArticoloDB(Item): bool

ProfiloDipendente

Il profilo dipendente è in grado di raggiungere la pagina magazzino attraverso il metodo "goMagazzino()".

Database Magazzino

La classe magazzino interagisce direttamente con la classe Database Magazzino che garantisce la persistenza dei dati.

CHAPTER

TWO

CODICE IN OBJECT CONSTRAINT LANGUAGE

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

2.1 Microservizio Task

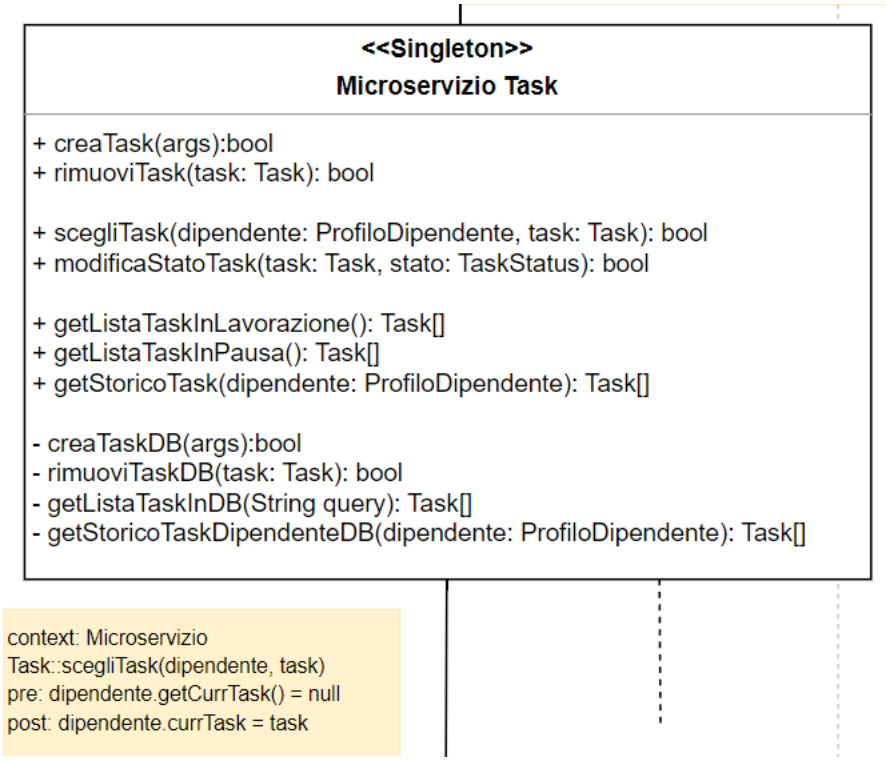

Diagramma delle classi con linguaggio OCL

Quando un dipendente sceglie una Task su cui lavorare non deve avere nessuna Task correntemente assegnata.

```
context: Microservizio Task::scegliTask(dipendente, task)
pre: dipendente.getCurrTask() = null
post: dipendente.currTask = task
```

2.2 Carrello

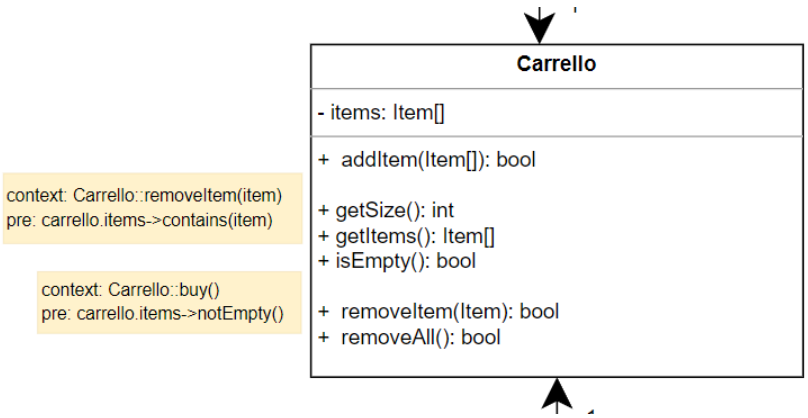

Diagramma delle classi con linguaggio OCL

Per rimuovere un articolo da un carrello è necessario che quell'articolo faccia parte del carrello.

```
context: Carrello::removeItem(item)
pre: carrello.items->contains(item)
```

Per comprare gli articoli del carrello è necessario che il carrello non sia vuoto

```
context: Carrello::buy()
pre: carrello.items->notEmpty()
```

2.3 Autenticazione

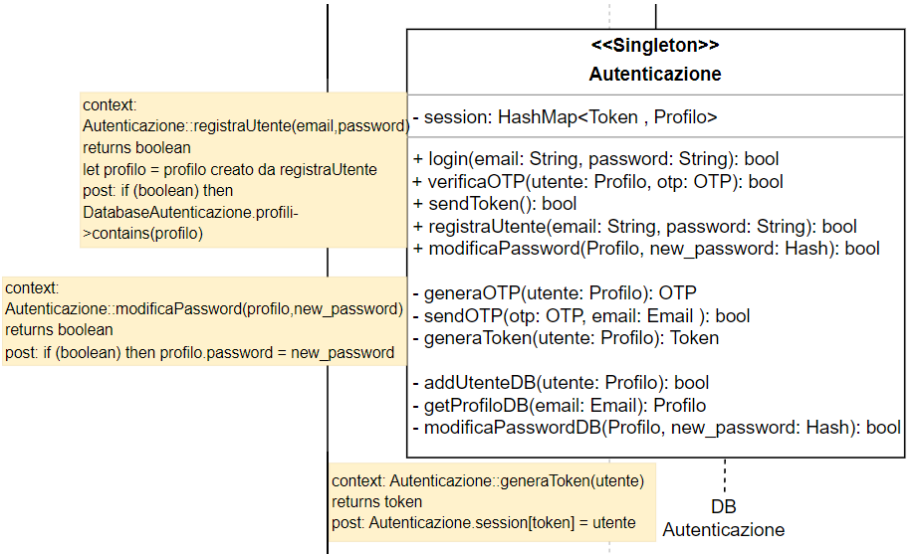

Diagramma delle classi con linguaggio OCL

una volta generato un Token utente, esso viene automaticamente inserito dentro alla session

```
context: Autenticazione::generaToken(utente) returns token
post: Autenticazione.session[token] = utente
```

Una volta modificata la password di un profilo utente, allora la password contenuta all'interno dello stesso dev'essere quella nuova

```
context: Autenticazione::modificaPassword(profilo,new_password) returns boolean
post: if (boolean) then profilo.password = new_password
```

Una volta registrato un utente, il suo profilo deve apparire nel database

```
context: Autenticazione::registraUtente(email,password) returns boolean
let profilo = profilo creato da registraUtente
post: if (boolean) then DatabaseAutenticazione.profili->contains(profilo)
```

2.4 Gestione Dipendenti

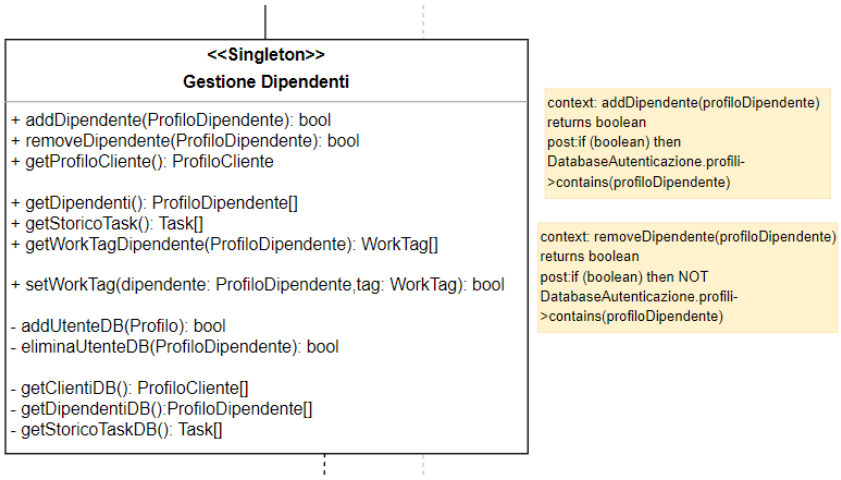

Diagramma delle classi con linguaggio OCL

Una volta che un profilo dipendente viene aggiunto, deve apparire all'interno del database

```
context: addDipendente(profiloDipendente) returns boolean
post:if (boolean) then DatabaseAutenticazione.profili->contains(profiloDipendente)
```

Una volta che un profilo dipendente viene rimosso, non deve più apparire all'interno del database

```
context: removeDipendente(profiloDipendente) returns boolean
post:if (boolean) then NOT DatabaseAutenticazione.profili->contains(profiloDipendente)
```

2.5 Negozio

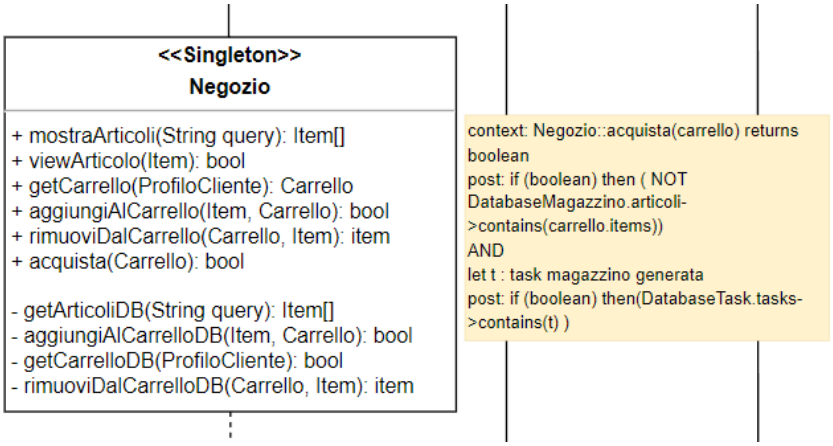

Diagramma delle classi con linguaggio OCL

Una volta acquistati gli articoli, il database non deve più contenerli. Inoltre dev'essere creata una Task magazzino.

```
context: Negozio::acquista(carrello) returns boolean
post: if (boolean) then ( NOT DatabaseMagazzino.articoli->contains(carrello.items))
AND
let t : task magazzino generata
post: if (boolean) then(DatabaseTask.tasks->contains(t) )
```

2.6 Assistenza, Riparazione, Feedback

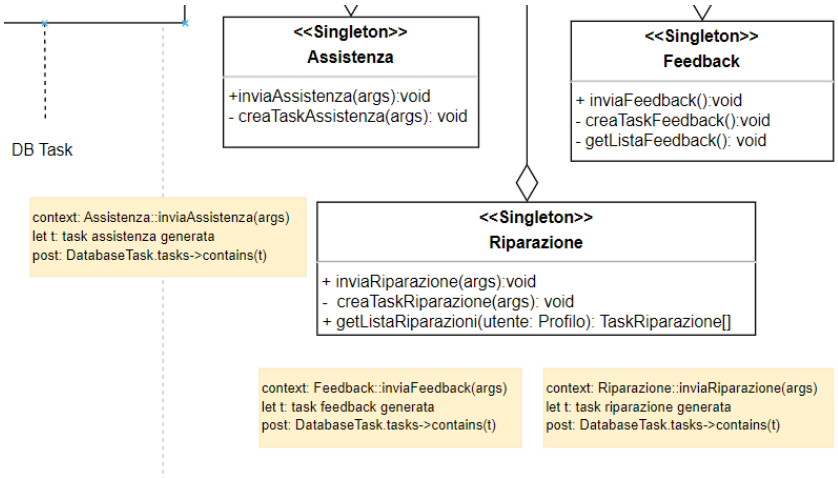

Diagramma delle classi con linguaggio OCL

Una volta inviata una richiesta di assistenza, riparazione o feedback, viene creata la corrispettiva Task e inserita nel database

```
context: Assistenza::inviaAssistenza(args)
let t: task assistenza generata
post: DatabaseTask.tasks->contains(t)

context: Feedback::inviaFeedback(args)
let t: task feedback generata
post: DatabaseTask.tasks->contains(t)

context: Riparazione::inviaRiparazione(args)
let t: task riparazione generata
post: DatabaseTask.tasks->contains(t)
```

2.7 ProfiloDipendente, ProfiloCliente, ProfiloManager

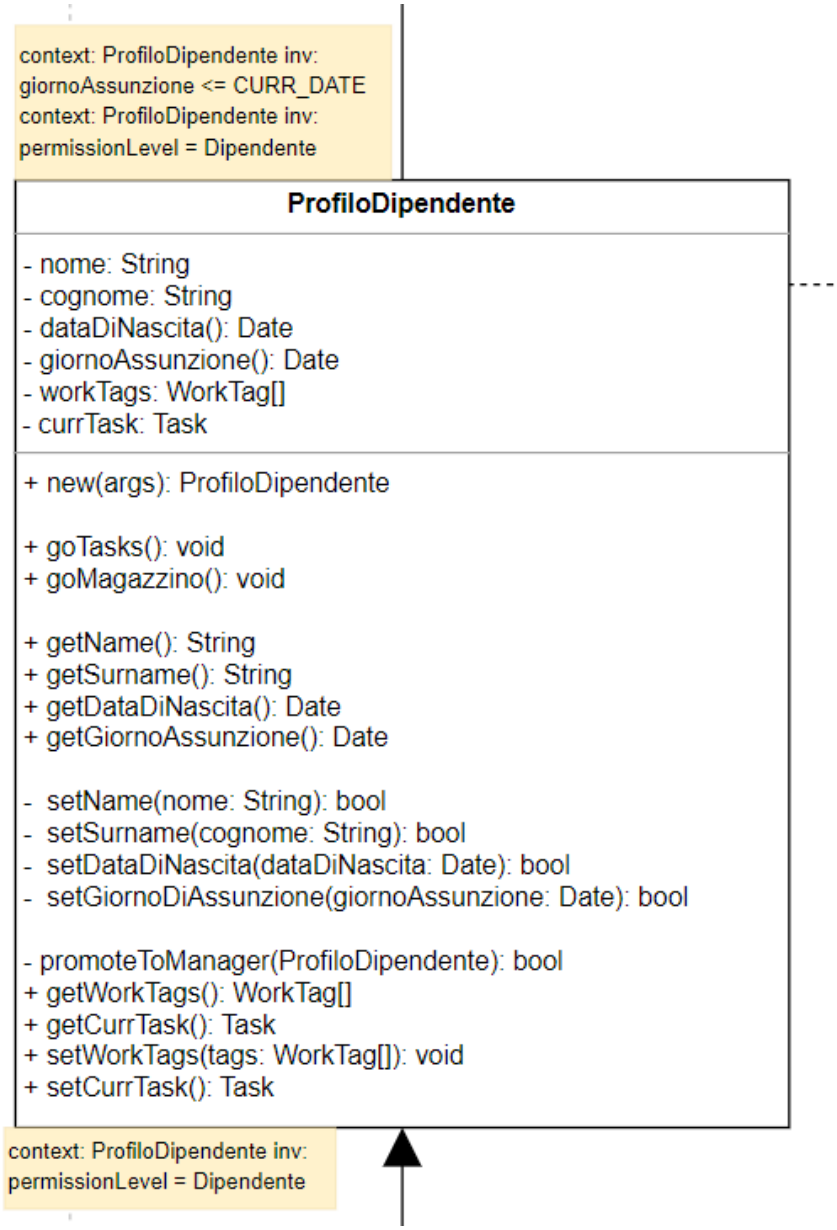

Diagramma delle classi con linguaggio OCL

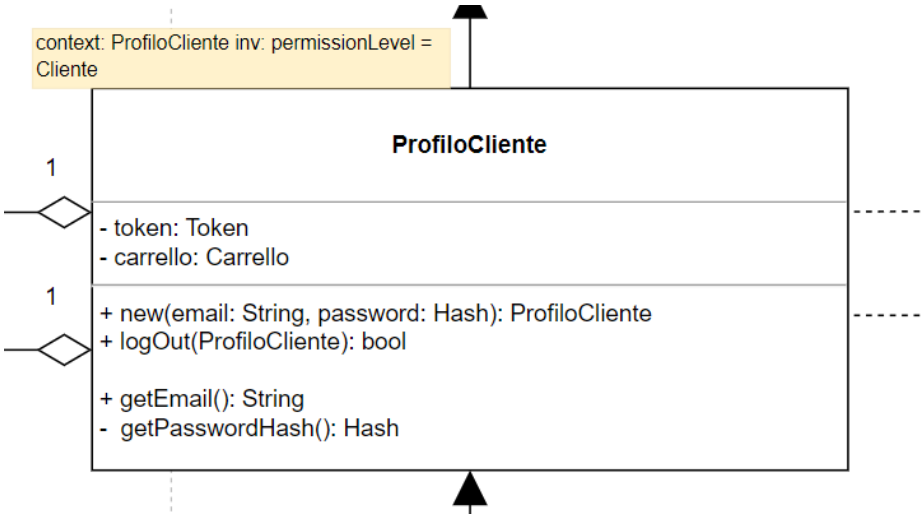

Diagramma delle classi con linguaggio OCL

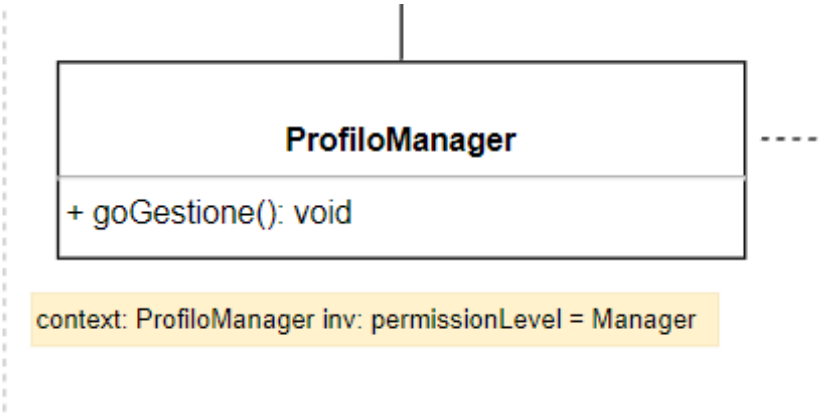

Diagramma delle classi con linguaggio OCL

La data d'assunzione del dipendente dev'essere inferiore a quella odierna

```
context: ProfiloDipendente inv: giornoAssunzione <= CURR_DATE
```

ProfiloDipendente, ProfiloCliente, ProfiloManager hanno i PermissionLevel rispettivi

```
context: ProfiloDipendente inv: permissionLevel = Dipendente
context: ProfiloCliente inv: permissionLevel = Cliente
context: ProfiloManager inv: permissionLevel = Manager
```

CHAPTER

THREE

DIAGRAMMA DELLE CLASSI CON CODICE OCL

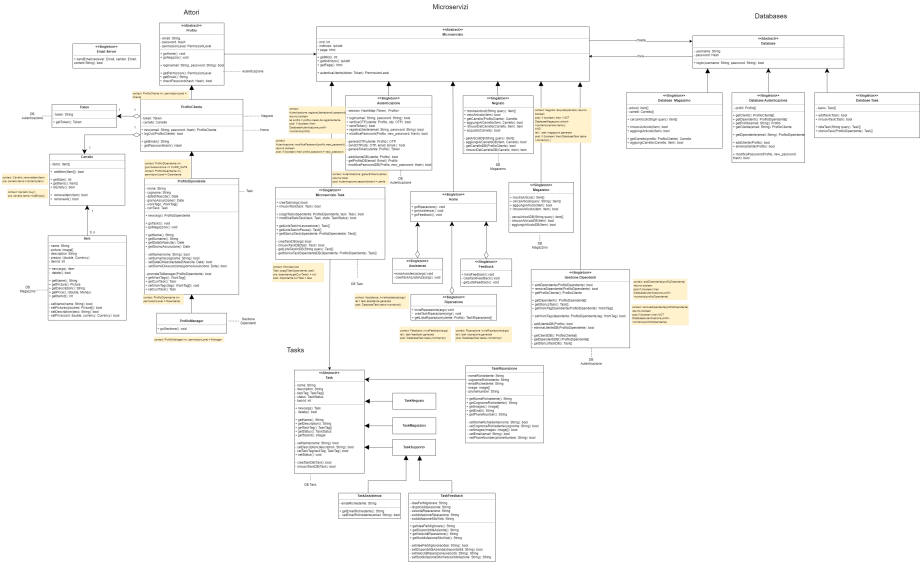

Diagramma delle classi con linguaggio OCL